Express Mail No. EL 588 610 921 US
Dated: February 6, 2002
Attorney Docket No.: N17-016

UNITED STATES UTILITY PATENT APPLICATION
(comprising 238 sheets of specification, claims, and abstract and 11 sheets of drawings)
in accordance with 35 U.S.C.§ 111(a) and 37 CFR 1.53(b)

of

Charles J. Northrup
a citizen of the United States

and

Franklin J. Angel
a citizen of the United States

both having a mailing address of
15 Spring Street, Suite 200
Princeton, NJ 08542

for

EXECUTION OF PROCESS BY REFERENCE TO DIRECTORY SERVICE

Gerry J. Elman
Attorney for Applicant
Reg. No. 24, 404
Stanley N. Protigal
Attorney for Applicant
Reg No. 28,657

ADDRESS ALL CORRESPONDENCE TO

ELMAN & ASSOCIATES
Customer no. 003775
20 West Third Street
P.O. Box 1969
Media, PA 19063

COPYRIGHT AUTHORIZATION

TITLE OF THE INVENTION

[0001] Execution of Process by Reference to Directory Service

BACKGROUND OF THE INVENTION

[0002] Field of the Invention

[0003] This invention relates to a network and provides a means for a user to provide a service, to consume a service, and to access and interact with a multiplicity of services.

[0004] Description of Related Art

[0005] The Internet and the World Wide Web have grown in size and complexity since inception. A common activity is to use a graphic rendering program such as Microsoft Internet Explorer, Netscape Navigator, Opera, or even Microsoft Word, to request and graphically render a Hypertext Markup Language (HTML) document. In requesting the HTML document, the user indicates a Uniform Resource Identifier (URI) to the graphic rendering process.

[0006] The following terms are defined in: "Hypertext Transfer Protocol -- HTTP/1.1, RFC 2616 Fielding, et al." One who is not skilled in the state of the art is encouraged to read the reference for clarity on the subject manner.

[0007] URI - Uniform Resource Identifier. The generic set of all names/addresses that are short strings that refer to resources.

[0008] URL - Uniform Resource Locator. An informal term (no longer used in technical specifications) associated with popular URI schemes: http, ftp, mailto, etc.

[0009] URN - Uniform Resource Name. A URN is an URI that has an institutional commitment to persistence, availability, etc. Note that this sort of URI may also be a URL. See, for example, PURLs. A particular scheme, urn:, specified by RFC2141 and related documents, intended to serve as persistent, location-independent, resource identifiers.

[0010] The "http" scheme is used to locate network resources via the HTTP protocol. This section defines the scheme-specific syntax

[0011] and semantics for http URLs.

[0012] http_URL = "http:" "//" host [":" port] [abs_path ["?" query]]

[0013] If the port is empty or not given, port 80 is assumed. The semantics are that the identified resource is located at the server listening for TCP connections on that port of that host, and the Request-URI for the resource is abs_path (section 5.1.2). The use of IP addresses in URLs SHOULD be avoided whenever possible (see RFC 1900 [24]). If the abs_path is not present in the URL, it MUST be given as "/" when used as a Request-URI for a resource (section 5.1.2). If a proxy receives a host name which is not a fully qualified domain name, it MAY add its domain to the host name it received. If a proxy receives a fully qualified domain name, the proxy MUST NOT change the host name.

[0014] By way of example, but not limitation, the user can enter an http schema URL such as:

[0015] http://www.gtlinc.com/products.html

[0016] In this example, the user is requesting the products.html document from the server given as www.gtlinc.com.

[0017] To retrieve the HTML document, the server must be running a Hypertext Transfer Protocol daemon (HTTPD) such as Apache from http://www.apache.org, or equivalent thereof. The HTTPD executes on a service provider system and listens for request on a port, typically port 80, which is a well-known, industry standard port, for the HTTP daemon. By using a standard port, a person can indicate to the Netscape Navigator, or equivalent thereof, to request an http document via a given Uniform Resource Location (URL). By having the standard port 80 used, anybody can request the URL since they do not have to worry about what port the HTTP Daemon is listening on. Otherwise, the user would have to indicate the desired port, such as http://www.gtlinc.com:399, where :399 indicates to connect on port 399. Using the industry standard port simplifies the data entry and the ability to access Hypertext Markup Language (HTML) documents.

[0018] A user of a computer system (or somebody on behalf of the user) pays for access to the Internet through an Internet Service Provider (ISP), such as AT&T WorldNet, America On-Line, or Microsoft Network. In a typical situation, the ISP frequently blocks request to port 80 on the user computer system to prevent the user from running a web site via an HTTP Daemon on their home computer, on the well known port 80. The user could provide the HTTP Daemon on a different port, such as port 399, but nobody would know to access that port unless the user published the port number. Even in publishing the port number, the enormous potential audience would unlikely see the advertisement.

[0019] Another challenge for the user accessing the Internet through an ISP, is that the ISP frequently uses Dynamic Addressing. In such circumstances, an

Internet Address is assigned only when the user connects to the Internet through the ISP. When the user disconnects, then the IP address will be reassigned to a different user. This poses a problem in publishing the alternative HTTP Daemon running on port 399 since the Internet Address changes each time the user access the Internet. Even cable modem providers frequently use dynamic internet addressing. In some cases though, a cable modem ISP may offer a dedicated Internet Address, but still frequently blocks port 80 on the user computer. In some cases, the ISP requires the user of the ISP service to enter an agreement wherein the user is precluded from running a service on port 80. Even if the user were to publish the current dynamic Internet address, they could only do so via publishing the physical address such as 190.190.83.2 and potentially the corresponding port. In any case, the user does not have a domain name associated with their computer such as gtlinc.com, wherein they could publish the domain name, which is easier for a prospect visitor to remember.

[0020] Although the problem of port blocking and dynamic Internet Address assignment frequently affects a user of an ISP service, similar challenges are faced by the industry in general. By way of example, but not limitation, a computer provider, a software provider, a tax service provider, a news service, a stock broker, a sales person selling goods or services, and others offering goods or services, are limited to providing the HTTP Daemon on port 80 because it is the industry standard port for the HTTP daemon. If any of the aforementioned wished to provide an alternative service on a port other than port 80, they would have to undertake a massive marketing campaign to educate potential visitors (users or businesses requesting information) on the particular port number.

[0021] The industry currently has products and services for providing directory services, but the directory service is generally limited to the enterprise within which the directory service is executing. By way of example, the Sun Microsystems iPlanet Directory Service is sold as a light weight directory access protocol for administering

directory services within the enterprise. Even at that, Sun marketing information indicates the iPlanet Directory Service as primarily for user administration within the enterprise. It does not provide a solution or function effectively for the global network. It does not provide a solution or function effectively for the Internet.

[0022] Industry members such as IBM, Microsoft, Hewlett Packard, SAP, and even Sun Microsystems have been indicating the Universal Definition Discovery Interchange (UDDI) as a means for providing information on service providers. The UDDI Specification, (available on-line at http://www.uddi.org) however, does not indicate registration of information such as other than port 80.

[0023] A more generalized solution for accessing and interacting with services provided on the Internet is needed.

[0024] It is therefore an object of this invention to provide methods and systems for accessing and interacting with a multiplicity of services.

[0025] The use of a service often will require payment for the services rendered. The standard method of providing credit card payment over the Web is viewed as insecure and tedious. A user completes a form displayed through the graphic rendering process and uses a pointing device such as a mouse to "click" on a graphical representation indicating to send the content of the user provided information to the service provider.

[0026] The Microsoft Corporation recently announced their Passport implementation wherein a user subscribes to the Microsoft Passport service, provides credit card information such as card type, card number, expiration date, card holder, billing address, and possible other information such as shipping address. The disadvantage of the Microsoft Passport implementation is that Microsoft controls

that information. By way of example, the subscriber payment information is maintained on a computer system administered by Microsoft. The data set that Microsoft maintains may be propagated to other servers as needed. While Microsoft claims the method to be secure, the disadvantage is that by having a centralized data set containing payment information for an enormous number of subscribers, would make that centralized data set a computer cracker's main target.

[0027] An alternative implementation is being proposed by Sun Microsystems under their Liberty Alliance consortium. Numerous members such as Mastercard, VISA, American Express, and others have signed up for the Liberty Alliance. The downside of the Liberty Alliance implementation is that as of today, the implementation is not yet defined. Furthermore, the indications are that they will still transmit credit card payment information to port 80 of the service provider providing the service (i.e., sale of service or goods is still a service). Sun Microsystems currently offers the Java Wallet, which is a family of products written in the Java programming language that are designed to enable secure commerce operations.

[0028] An alternative payment mechanism is provided by PayPal, which is used quite frequently for auction sites such as www.ebay.com. The PayPal implementation, however, requires PayPal to act in the capacity of a credit card merchant. Therefore a buyer provides PayPal with credit card information and PayPal charges the credit card and receives payment. PayPal then credits the seller's account with the appropriate amount. A second disadvantage is that PayPal charges a transaction fee which is then deducted from the seller's amount. A third disadvantage is that both the buyer and the seller must provide account information, which is then maintained by PayPal.

[0029] It is understood that a user of a computer system could cause a process to execute wherein the process can provide payment information to a requesting process. The disadvantage is that there is no mechanism for verifying whom the requesting process is executing on behalf of. In this case, the user process could provide payment information to anybody, including a computer hacker, and thus is unacceptable.

[0030] It is therefore another object of this invention to provide methods and systems for payment of services.

[0031] In the current state of the computing industry, a user who desires to access a web page, but, who does not know the corresponding URI, must use a browser such as Microsoft Internet Explorer to visit a search engine such as Yahoo or Google and submit keywords to query for pages satisfying their request. The user is then presented with one or more URIs and text descriptions of the content at the URI. The user can then "click" on one of the URIs satisfying the request. The corresponding HTML document is then retrieved and rendered for the user to see. A disadvantage is that the user must undergo a two-step approach. First, the user must visit Google, enter the terms, and then "click" on the desired URI.

[0032] It is therefore another object of this invention to provide methods and systems for simplifying connections.

[0033] An alternative is provided by RealNames. RealNames allows a corporation, such as Global Technologies Ltd., Inc., to register a keyword GTL so that when a user enters GTL as the desired site, the RealName would be translated to http://www.gtlinc.com. The challenge, of course, is that the user must know the keyword.

SUMMARY OF THE INVENTION

[0034] According to the present invention, a method for using a service in a computer network a first software component executes on a first computer. The first software component registers as a service with a directory service process executing on a second computer, and the directory service process creates a registration for the first component of software. A second component of software executes on a third computer and communicates to the directory service process, a request to access and interact with the first software component. The directory service process responds by locating the registration entry for the first component of software, and facilitates communication with the first component of software on behalf of the second component of software.

BRIEF DESCRIPTION OF THE DRAWINGS AND LISTINGS

[0035] Figure 1 is a diagram of a computer network communicating according to the present invention.

[0036] Figures 2-7 are flow charts of the operation of the present invention.

[0037] Figure 2 is a flowchart of a directory service connection service.

[0038] Figure 3 is a flowchart of a directory service use.

[0039] Figure 4 is a flowchart of a service provider registration.

[0040] Figure 5 is a flowchart of a service registration.

[0041] Figure 6 is a flowchart of a consumer registration.

[0042] Figure 7 is a flowchart of a consumer request for service.

[0043] Figures 8-13 are diagrams showing the communications relationships of different types of data providers in accordance with the present invention.

[0044] Figure 8 is a schematic block diagram of connectivity depicting horizontal partition by category.

[0045] Figure 9 is a schematic block diagram of connectivity depicting horizontal partition by provider.

[0046] Figure 10 is a schematic block diagram of connectivity depicting horizontal partition by activity.

[0047] Figure 11 is a schematic block diagram of connectivity depicting horizontal partition by cost.

[0048] Figure 12 is a schematic block diagram of connectivity depicting horizontal partition by protocol.

[0049] Figure 13 is a schematic block diagram of connectivity depicting horizontal partition by entity type.

[0050] Figures 14-16 are diagrams depicting data transfer provided by a directory service.

[0051] Figure 14 is a diagram depicting a sample TDS with three service directories according to the present invention.

[0052] Figure 15 is a diagram depicting a sample environment with five systems sharing TDS information according to the present invention.

[0053] Figure 16 is a diagram depicting a sample TDS configuration as applied to directories provided through the Sun Solaris 2.7 operating system according to the present invention.

[0054] The program listings are as follows:

[0055] Program Listing 1.1 source code listing of one implementation for the replacement recv function

[0056] Program Listing 2.0 Engine Service engine.c

[0057] Program Listing 2.1 Engine Service getnvpair.c

[0058] Program Listing 2.2 Engine Service authorize.c: placeholder authorization service

[0059] Program Listing 2.3 Engine Service input.c: placeholder input service

[0060] Program Listing 2.4 Engine Service postprocess.c: placeholder postprocess service

[0061] Program Listing 2.5 Engine Service preprocess.c: placeholder preprocess service

[0062] Program Listing 2.6 Engine Service process.c: placeholder process service

[0063] Program Listing 2.7 Engine Service response.c: placeholder response service

[0064] Program Listing 2.8 Engine Service readline.c:

[0065] Program Listing 2.9 Engine Service wait_read.c

[0066] Program Listing 2.10 Engine Service - peek.c

[0067] Program Listing 2.11 Engine Service peek_c.c

[0068] Program Listing 2.12 Engine Service main.c

[0069] Program Listing 2.13 Engine Service - Makefile

[0070] Program Listing 2.14 Engine Service - engine.mk

[0071] Program Listing 2.15 Engine Service - dummy.mk

[0072] Program Listing 2.16 Engine Service - engine.conf

[0073] Program Listing 3.0 authentication service - authenticate.c

[0074] Program Listing 3.1 Authentication Service - log.h

[0075] Program Listing 3.2 Authentication Service - tds2.h

[0076] Program Listing 3.3 Authentication Service - makefile

[0077] Program Listing 3.4 authentication Service - authenticate.conf

[0078] Program Listing 4.1 - Thread Directory Service - tds3.c

[0079] Program Listing 4.2 - Thread Directory Service - ste.c

[0080] Program Listing 4.3 - Thread Directory Service - log.c

[0081] Program Listing 4.4 Thread Directory Service - ic.c

[0082] Program Listing 4.5 thread directory service - set_blocking.c

[0083] Program Listing 4.6 thread directory service - set_nonblocking.

[0084] Program Listing 4.7 thread directory service - Makefile

[0085] Program Listing 5.0 fopen service - fopen.c

[0086] Program Listing 6.0 fscanf service - fscanf.c

[0087] Program Listing 7.0 fclose service - fclose.c

[0088] Program Listing 8.0 caps service caps.c

[0089] Program Listing 9.1 generic front end loader service gfel.c

[0090] Program Listing 9.2 generic front end loader service client_gl.c

[0091] Program Listing 9.3 generic front end loader service client_gl2.c

[0092] Program Listing 9.4 generic front end loader service gl3.c

[0093] Program Listing 10.1 thread connection service - talk2.c

[0094] Program Listing 10.2 thread connection service - participant.c

[0095] Program Listing 10.3 thread connection service - tcp_accept2.c

[0096] Program Listing 10.4 thread connection service - tcp_connect.c

[0097] Program Listing 10.5 thread connection service - tcp_listen.c

[0098] Program Listing 11.1 supporting functions - reaper.c

[0099] Program Listing 12.1 supporting service - cat_service.c

[0100] Program Listing 12.2 supporting service - echo_service.c

[0101] Program Listing 12.3 supporting service - daytime_service.c

[0102] Program Listing 12.4 supporting service - ksh_service.c

[0103] Program Listing 12.5 mail service - mail_service.c

[0104] Program Listing 13.1 TDS supporting functions - tds_query_p.c

[0105] Program Listing 13.2 TDS supporting functions - tds_register_p.c

[0106] Program Listing 13.3 TDS supporting functions - getdtscinfo.c

[0107] Program Listing 13.4 TDS supporting functions - tds.c

[0108] Program Listing 14.0 process function - cps.c

[0109] Program Listing 14.1 process function - cps2.c

[0110] Program Listing 14.2 process function - cps3.c

[0111] Program Listing 15.0 stateful service - main.c

[0112] Program Listing 15.1 stateful service - tcp_accept.c

[0113] Program Listing 15.2 stateful service - tcp_listen.c

[0114] Program Listing 15.3 stateful service - getaddrinfo.c

[0115] Program Listing 16.1 - File SERVICES1 Service prototype table.

[0116] Program Listing 16.2 - File SERVICES2 Service prototype table.

[0117] Program Listing 16.3 - File SERVICES3 Service prototype table.

[0118] Program Listing 16.4 - Command line to generate data dictionary from prototype table

[0119] Program Listing 16.5 - Generated Data Dictionary

[0120] Program Listing 16.6 - Services2 prototype table

[0121] Program Listing 16.7 - Generated Data Dictionary for Services2

[0122] Program Listing 16.8 - Providers prototype table

[0123] Program Listing 16.9 - Providers generated data dictionary

[0124] Program Listing 16.10 - Cymbal instructions to insert record

[0125] Program Listing 16.11 - Cymbal instructions to report registration entry information

[0126] Program Listing 16.12 - Global Definitions

[0127] The Architecture

[0128] The Internet is a network linking computer systems together and communicating via a standard protocol. A computer network is simply a collection of autonomous computers connected together to permit sharing of hardware and software resources, and to increase overall reliability. The qualifying term "local area" is usually applied to computer networks in which the computers are located in a single building or in nearby buildings, such as on a college campus or at a single corporate site. When the computers are further apart the term "wide area network" may be used.

[0129] As computer networks have developed, various approaches have been used in the choice of communication medium, network topology, message format, protocols for channel access, and so forth. Some of these approaches have emerged as

de facto standards, but there is still no single standard for network communication. The Internet is a continually evolving collection of networks, including Arpanet, NSFnet, regional networks, local networks at a number of university and research institutions, a number of military networks, and increasing, various commercial networks. The protocols generally referred to as TCP/IP were originally developed for use through Arpanet and have subsequently become widely used in the industry. The protocols provide a set of services that permit processes to communicate with each other across the entire Internet.

[0130] A computer can be a mainframe, minicomputer, microcomputer, or any of a number of other computing devices. In the case of the present invention, the computer should be able to communicate with the outside world. Therefore, for example, a first generation microwave oven controller using a Z-80 chip would not be able to use the invention, but it is conceivable that providing a communications capability to a microwave controller would enable it to use the invention. A number of different computing devices are able to communicate with the outside world while computing. Such devices include set top boxes, PDAs (personal digital assistants), and cellular phones using CDMA or similar technologies.

[0131] Likewise, a server is traditionally at a fixed location; however it is possible to provide a server in any of a number of forms. The server can be running as a client of another server and in fact it is often the case that a computing device may be a client to another device which functions as a host, and yet perform server functions for that other device.

[0132] A model for network architectures has been proposed and widely accepted. It is known as the International Standards Organization (ISO) Open Systems Interconnection (OSI) reference model. The OSI reference model is not itself a network architecture. Rather it specifies a hierarchy of protocol layers and

defines the function of each layer in the network. Each layer in one computer of the network carries on a conversation with the corresponding layer in another computer with which communication is taking place, in accordance with a protocol defining the rules of this communication. In reality, information is transferred down from layer to layer in one computer, then through the channel medium and back up the successive layers of the other computer. However, for purposes of design of the various layers and understanding their functions, it is easier to consider each of the layers as communicating with its counterpart at the same level, in a "horizontal" direction. (See, e.g. The TCP/IP Companion, by Martin R. Arick, Boston: QED Publishing Group 1993, and U.S. Pat. No. 5,159,592. These, and all patents and publications referenced herein, are hereby incorporated by reference.)

[0133] The lowest layer defined by the OSI model is called the "physical layer," and is concerned with transmitting raw data bits over the communication channel. Design of the physical layer involves issues of electrical, mechanical or optical engineering, depending on the medium used for the communication channel. The second layer, next above the physical layer, is called the "data link" layer. The main task of the data link layer is to transform the physical layer, which interfaces directly with the channel medium, into a communication link that appears error-free to the next layer above, known as the network layer. The data link layer performs such functions as structuring data into packets or frames, and attaching control information to the packets or frames, such as checksums for error detection, and packet numbers.

[0134] The Internet Protocol (IP) is implemented in the third layer of the OSI reference model, the "network layer," and provides a basic service to TCP: delivering datagrams to their destinations. TCP simply hands IP a datagram with an intended destination; IP is unaware of any relationship between successive datagrams, and merely handles routing of each datagram to its destination. If the destination is a

station connected to a different LAN, the IP makes use of routers to forward the message.

[0135] The basic function of the Transmission Control Protocol (TCP) is to make sure that commands and messages from an application protocol, such as computer mail, are sent to their desired destinations. TCP keeps track of what is sent, and retransmits anything that does not get to its destination correctly. If any message is too long to be sent as one "datagram," TCP will split it into multiple datagrams and makes sure that they all arrive correctly and are reassembled for the application program at the receiving end. Since these functions are needed for many applications, they are collected into a separate protocol (TCP) rather than being part of each application. TCP is implemented in the "transport layer," namely the fourth layer of the OSI reference model.

[0136] Except as otherwise is evident from the context, the various functions of the present invention reside above the transport layer of the OSI model. The present invention may be used in conjunction with TCP/IP at the transport and network layers, as well as with any other protocol that may be selected.

[0137] The OSI model provides for three layers above the transport layer, namely a "session layer," a "presentation layer," and an "application layer," but in the Internet these theoretical "layers" are undifferentiated and generally are all handled by software.

[0138] Internet Firewall

[0139] A security system placed between the Internet and an organization's network (such as a LAN) to provide a barrier against security attacks. Internet firewalls typically operate by monitoring incoming and/or outgoing traffic to/from the

organization's network, and by allowing only certain types of messages to pass. For example, a firewall may be configured to allow the passage of all TCP/IP traffic addressed to port 80, and to block all other traffic. For more information of Internet Firewalls, see Chapman and Zwicky, Building Internet Firewalls, O'Reilly publishing, 1995 (ISBN 1-56592-124-0).

[0140] Computer systems having access to the Internet, can have a dynamic Internet Address assigned to them. The Internet Firewall can be configured to perform network address translation as defined in "Network Working group Request for Comments 1631, and Request for Comments 3022."

[0141] A computer system having access to the Internet can be assigned a private Internet Address, as defined in Request For Comments 1597.

[0142] Component of Software

[0143] A basic principle of the invention is that of a component of software. The term component of software is deliberately chosen to indicate that less then an executable program may be used. By way of example, but not limitation, a component of software can be:

- an executable program
- an executable program linked with shared libraries, dynamic link libraries, or other such libraries as would be provided for in an embodiment
- an object as one would understand in using remote procedure call
- an object as one would understand in using the Microsoft Component Object Model or other such industry standard

- a dynamically loadable module such as a module in a shared library (also called dynamic link library on Microsoft Windows) or other such library as defined by the operating system or embodiment
- a function that is called by a dynamically loadable library initialization function, such as occurs with the use of a Microsoft's Windows DLL. In such cases, a DllMain function may be called when a thread (either a process, or a thread created by the process) attaches to the library. Initialization functions are also accessible through KornShell and other such processes. The initialization function may therefore perform the functionality required of the component of software
- a software assembly as defined in the Microsoft C# Language
- a builtin function of a shell program such as the KornShell
- a function of an interpretive language processing element, such as a KornShell function, shell function, or a perl function.
- a shell script as defined by a shell program such as the KornShell or other interpretive language processing element.
- a script that is interpreted by another process such as that which is used by BASIC, Kornshell, Csh, Tcsh, Perl, Tcl/Tk, or other such interpreter
- a module which is then linked into an executable with a just-in-time compiler
- a byte stream which is communicated to an interpreter such as that which is available with KornShell, Java
- a data stream which is communicated to an interpreter process

[0144] Note that when used with the invention, the component of software may require the use of a generic front end loader process that initializes an address space. By way of example, but not limitation such a generic front end loader could:

- accept command line parameters identifying the component of software to be used, or

- determine such information by accessing a configuration, or
- accessing a memory location accessible to the generic front-end loader, or
- communication with a second process providing such information, or
- accessing and interacting with a directory service process, or
- accessing and interacting with a component of software to determine such information, or
- communicating with a second process to determine such information, or
- use inter-process communications to determine such information, or
- use intra-process communications to determine such information, or
- use operating system interfaces providing such information, or
- use an application programming interface to determine such information, or
- use a combination of the above to determine such information.

[0145] Note that when the component of software is provided by a data stream interpreted by an interpreter, then the data stream may require a local process to communicate with an accessible process in order to facilitate the data stream. By way of example, but not limitation, such a data stream may be communicated from an Internet Address and Port as one would understand when using the socket application programming interface, or equivalent thereof. Alternative network Application Programming Interfaces can be used (See the discussion on communications for examples). Such an implementation would require connecting to the process at the specified network (which could include an Internet Address and port), possibly communicating a request to the connected process, and receiving a response wherein the response communication includes the data stream. Alternatively, by way of example, such a data stream may be communicated from a process accessible through communications over the Internet wherein the process is

defined by a Universal Resource Location (URL) as in http://www.gtlinc.com/proc/stream or equivalent thereof.

[0146] A device driver can be used. Either one provided through the operating system interfaces, or, one provided by an application operating environment such as the AST ToolKit. By way of example, but not limitation, an implementation can use an open system call to open a device such that by accessing and interacting with the device, information such as that required for facilitating the methods, can be achieved. By way of example, a process issues:

[0147] fp=fopen("directory service", "rw");

[0148] The process opens a device called directory service. As this may not be an operating system device, the fopen implementation determines how to access and interact with the device based on the device name specified. See function call and system calls for details on implementing augmented functions.

[0149] A component of software can be installed on the computer system, or accessible to the computer system through the network. A user, such as a consumer, or a service provider, can cause the software to be installed. This can include the use of software downloaded from the network, as well as software that is preinstalled on the computer system as purchased, or software that is installed during the installation of the operating system or component thereof. The component of software downloaded from the network may require an installation process to be executed, which then installs on the computer such that it can be executed. By way of example, but not limitation, a first component of software downloaded can be compressed and require decompression, resulting in an executable that then installs one or more components of software on the computer system. Examples would

include such techniques as downloading an InstallShield package, a Java component, a C# Assembly or other such techniques as known in the industry.

[0150] One or more programming languages and programming techniques can be used to create various embodiments of the invention, and the invention can be implemented on various operating systems such as AIX, BSD, Linux, HP-UX, Solaris, UNIX, IRIX, OpenEdition, UnixWare, and Windows.

[0151] A component of software can provide a service for a daemon process listening on a particular network endpoint, such as Internet Address and port (i.e. 192.127.0.3 port 80). In such cases, the information communicated to the daemon process will be used by the daemon process to cause the service to be executed. According to US Patent 5,850,518, the service can be dynamically loaded, or can be executed in a manner in which the daemon process connects to the service via a communication link. Such cases may be necessary to provide the desired functionality.

[0152] Program Listing 15.0 through 15.3 provide an embodiment using a main program that accepts command line parameters indicating the type of primitive to use, the internet address and port, and the name of the service to load. The service is dynamically loaded from the libservices.so.1.0 library. Each time a connection is received, the service is invoked.

[0153] Application Service

[0154] An application is said to provide a primary service. The application may also offer one or more minor services. The primary service, along with any minor services, collectively constitute the application service. By way of example, an application such as the Netscape Communicator can provide a primary service of

graphically rendering HTML documents. A minor service offered by the Netscape Communicator is a Messenger for administering (such as creating, sending, receiving, deleting, cataloging, viewing, forwarding, editing) electronic mail. A second minor service offered by the Netscape Communicator is a Composer for creating new HTML documents or editing existing documents. One skilled in the state of the art would understand that the a first user of an application could perceive the application as providing a primary service that is different from a second user of the same application.

[0155] Minor Service

[0156] A minor service provides some functionality towards the overall application service. The Minor Service is implemented through a component of software. When used in an active context, it is understood that the term Minor Service refers to the process executing the component of software. When used in the inactive context, the minor service refers to the component of software. Thus one would understand that a minor service is provided by a component of software and when the application requires interaction with the minor service, then the minor service is executing.

[0157] Service

[0158] A service is provided for by a component of software. A service may be a minor service, or a primary service. A service can be a primary service of a first application service, and a minor service of a second application service. A service can be a service to itself. By way of example, a service can be implemented as a first process which then issues a fork() system call to create a child process.

[0159] In standard UNIX environments, its it standard coding technique to create a daemon process listening for requests for services on a particular Internet

Address and port. When a client connects to the specified port, then the daemon process will typically accept the connection, and then issue a fork function call. The fork function creates a child process. The original daemon process, called the parent process, remains executing. The child process typically closes its standard input, standard out, and standard error file descriptors. The child process then duplicates the file descriptor (or handle) associated with the accepted connection, as the new standard input, standard output, and standard error file descriptors. The child process then typically issues an exec function call. The exec function call overlays the image of the current process with a new image of a new executable program to be executed. The child process typically performs whatever action is necessary, and then exits.

[0160] There are cases, however, where the process providing the service may need to stay executing even after responding to the first requesting process. Different methods can be used. One method is for the first process to accept the connection, perform the desired service, and respond to the requesting process. In this manner, whatever state changes where made to the first process remain intact, and are available to subsequent processes. A second method is for the first process to create the child process, and to have the child process remain executing. In this manner, the changes made to the state of the child remain intact. For subsequent requesting processes to gain access to such state information, the child process provide means to permit the subsequent requesting process to access and interact with the child, which may include having the child connect to the requesting process, or, having the requesting process connect to the child, or both. An example of where such state information is useful to retain is when the service is to provide a function or system call on behalf of a requesting process. There are cases where the result of the function or system call must be retained by the service and accessible to subsequent requesting processes (which could be the same requesting process later accessing and interacting with the service). By way of example, a first requesting

process sends a request to a service to perform a file open function call. The service perform the open function call and has associated therewith a file descriptor (or handle). The service provides the results to the requesting process. The requesting process then disconnects. A requesting process later accesses and interacts with the service, providing the service with the previous response indicating the results of the open function call. The requesting process provides a request indicating the service is to read a string from the file descriptor. The service, still having the file descriptor open, performs the read and returns the results thereof to the requesting process.

[0161] Application Process

[0162] The term application process, as used in this document, refers to the overall computer representation of the application service. In this definition, the term application process is defined to incorporate all processes of various "weight" including, but not limited to, heavy weight, medium weight, and light weight processes relating to the application service. A heavy-weight process executes in its own address space, whereas medium-weight and light-weight processes may execute within the same address space. The application process may constitute one or more of these processes. Each of these processes is said to have a thread of execution.

[0163] A thread, in this context, represents an execution stream of the application process. The notion of a thread can be provided by the underlying operating system, referred to as kernel-supported threads, or can be provided at the application level, referred to as user-level threads, or can be a mixture of the two. For the purposes of this description, these will collectively be referred to as threads. Note that in a distributed environment, one or more of these threads may be executing on a remote computer system.

[0164] The application process may be confined locally to the computer system on which the application process was initially started, or may have its execution

threads distributed among various computer systems accessible to the computer system on which the application process was initially started.

[0165] When a user of the computer system requests to execute an application, a corresponding program is loaded into the computer's memory and a single thread of execution begins. This initial thread may then create additional threads on the local computer system, or possibly on a remote computer system, such as that which would occur with remote procedure call implementations, Microsoft COM, Microsoft DCOM, or other such industry standard techniques.

[0166] The creation of a new thread requires the starting point of the new thread to be specified. In procedural computer languages, for example, this would require the requesting thread to specify the address of the procedure to begin as a new thread.

[0167] Communication Devices

[0168] A computer system includes a communication device. By way of example, but not limitation, a communication device can be a modem, a network card, a RFC device, an infrared device, an optical device, a wireless device, a device connecting the computer to a public switching system device, such as that provided for by a telephone carrier, a T1 connection or equivalent thereof, or any such device for the purpose of facilitating communication between one or more computer systems. All such devices are referred to as communication devices.

[0169] A process can listen on a communication device, awaiting a communication. By way of example, but not limitation, a process can be considered a daemon process, such as that provided by inetd on a UNIX implementation, or other such process, and await a communication. When a communication is received, the

process can accept the connection and then send communications, receive communications, or otherwise interact with the communication as appropriate.

[0170] A process that is listening on a communication device generally has a file descriptor open associated with the device. Certain embodiments, such as that with the Microsoft Windows operating system environment, can alternatively use a socket handle to listen on the device. Note, however, that with the U/WIN environment available from Global Technologies Ltd., Inc., the code would refer to a file descriptor that is then translated to a handle for the underlying operating system.

[0171] When a process accepts a connection, the process can cause a second process to begin executing. Alternatively, the second process may already be executing. In either case, the first process can inform the second process of the file descriptor, or handle, that the first process accepted the communication connection on. Various techniques are available to implement this. By way of example, but not limitation, the first process can cause the second process to be created and the file descriptor, or handle, can be inherited by the second process. Alternatively, the first process can open the second process and duplicate the handle from the first process to the second process. Alternatively, the second process can open the first process and duplicate the handle from the first process to the second process. Alternatively, the first process can use file descriptor passing techniques to pass the file descriptor or handle to the second process.

[0172] Communication

[0173] Interprocess, Intraprocess, and network communications are supported. Communication from a first process executing on a first computer to a second process executing on a second computer requires the use of a communication device. The

operating system typically provides interfaces for communication connectivity and synchronization in using such communication devices. The operating system interfaces generally provide for a connect/send/receive/disconnect capability. Note, though, that a device can be referenced with equivalent functionality using an open/write/read/close interface, or some other interface as provided for by intermediary components of software providing equivalent functionality.

[0174] By way of example, but not limitation, the socket application programming interface can be used to facilitate communicate. On the Microsoft Windows operating system, equivalent Win32 Application Programming Interfaces can be used.

[0175] It is expressly understood that when a first process communicates with a second process, the communication may be sent by the first process on a first computer to a second process on a second computer and that such communication may be sent through intermediary computer systems on the network. Thus the communication from the first computer may be processed by one or more intermediaries before arriving at the final destination which is the second process.

[0176] It is expressly understood that when a first process communicates with a second process, the communication can be sent by the first process to a process executing on a second computer, and that this process can cause the communication to be made available to the second process. By way of example, but not limitation, the phrase "a first process sends a communication to a second process" can be understood as the first process sends a communication to a daemon process which receives the communication, causes the second process to begin executing, and causes the communication to be accessible to the second process. By way of example, but not limitation, the phrase "causes" can be interpreted as the process provides the second process with the file descriptor or handle, or, the process receives the

communication and uses interprocess or intraprocess communications to make the communication available to the second process.

[0177] As provided for by US Patent 5,850,518 patent, a process can create a thread to perform the communication. By way of example, but not limitation, a first process can create a reader thread to receive a communication from a second process. When a message is received by the reader thread, the first process is notified and can access and interact with the message.

[0178] Various forms of encryption, message scrambling, or other such techniques can be used by the implementation to add additional layers of security as required by the implementation.

[0179] Content and Format

[0180] The term communicate implies content. It is further understood that the format of the content of the communication can be defined by the embodiment. By way of example, but not limitation, formats such as HTML, SGML, XML, schema information, data type information, name value pairs, text, or even components of software fabricated to convey the information. A shell script, for example, can have variable names and values to convey information. The only limitation is that the participants in the conversation must have a method to communicate the necessary information. This may, for example, include the use of various filters or translation services to transpose the communicated content from a first format to a second format, and possibly from the second format back to the first format. A multiplicity of formats may be used along the path as the communicated content moves along the network.

[0181] One skilled in the state of the art would understand that content could be expressed according to rules of grammar. For example, a scripting language such as KornShell, or Perl, or Tcl, or Tk, employ particular grammatical rules. It is understood that the content can be formatted according to a language's grammatical rules.

[0182] Furthermore, content can be filtered through various filtering techniques as defined by the implementation.

[0183] Furthermore, content can be verified through various verification techniques as defined by the implementation. By way of example, but not limitation, the verification can be implemented through one or more of:

- the use of XML facets
- the use of components of software such as that provided for by the Daytona Data Management system
- the use of a binding service, such as that provided for by the methods of US Patent 5,850,518
- the use of industry standard protocols
- the use of industry standard specifications.

[0184] Protocols

[0185] Communication implies the use of a protocol. A protocol defines a set of rules for communication. Protocols such as TCP, HTTP, FTP, computer mail protocols, application defined protocols, industry standard protocols, proprietary protocols, and the likes can be used. Once skilled in the state of the art would understand that various protocols could be developed in the future which can also be used for such communication. Furthermore, a multiplicity of protocols may be used as required. By way of example, but not limitation, protocols such as SOAP (Simple

Object Access Protocol) can be used in conjunction with transport protocols such as HTTP. From the standpoint of the invention, all such protocols are contemplated for and collectively referred to as a protocol.

[0186] Consumer Service

[0187] The term consumer is meant as a consumer of a service. The service consumed can be an on-line service such as banking, electronic commerce, data acquisition, news reports, a service describing something of interest, changes to a web site, changes to a catalog, changes to what is available on-line, or even an online service such as that provided by an Internet Service Provider. Regardless, though, the service is provided by at least one component of software. A person, acting as a consumer, can also provide a service and such a service is referred to as a consumer service. In such cases, the consumer service is provided by a component of software.

[0188] A consumer causes a component of software to be installed on the computer system wherein the component of software provides a consumer service. Alternatively, the component of software can be pre-installed by a provider of such computing device as one may anticipate when purchasing a computer from a provider such as Compaq, Dell, or Gateway. Alternatively a component of software can be downloaded from the network which implies the use of transferring the component of software from a first computer to a second computer, the second computer representative of the computer system being used by the consumer.

[0189] Registry

[0190] The term registry is understood to imply a collection of related data. The term service directory could be used as well. The embodiment can use a database, a data management system such as the Daytona Data Management System

from Global Technologies Ltd., Inc., a directory service, an ascii text file, a binary file, an indexed file, an industry standard method of organizing data, a method for administering data as provided for by an operating system, or other such techniques as would be understood in the state of the art, to facilitate the administration and administrative functions required. Such administrative functions can include one or more of collecting, organizing, accessing, interacting, verifying, replicating, or indexing of such information. A minimum administrative functionality set should include the ability to register, query, and delete. Additional administrative functionality would include the ability to change, update, or otherwise modify existing data. Within this specification, a directory service constitutes the application service for administering the data in the registry. When implemented with the Daytona Data Management System, a multiplicity of individual programs, libraries, applications can collectively constitute the directory service.

[0191] In a preferred embodiment, the Daytona Data Management System would be used instead of a commercial database system such as Oracle. The distinction is that Daytona provides full database capability in the development environment, and supports a runtime environment without the capability to define or add new tables and new schemas. A Daytona runtime environment has a significantly lower cost then comparable database systems such as Orcale or Informix, and does not require the customer to hire a database administrator. The Daytona system is specialized for run time applications needing data management, without the overhead of a Oracle or Informix.

[0192] Multiple registries can be used, and the registries may reside on different computers of the network. In one sense, this can be used to provide collections of services based on geographic areas. By way of example, a first registry contains entries representative of service providers providing service only within the state of New Jersey. A second registry contains entries representative of service

providers providing service only within the state of New York. One skilled in the state of the art would understand that both registries could reside on a single server located in Connecticut, or on a first server in New Jersey and a second service in New York.

[0193] The organization of the data within the registry can be defined by a schema, as one skilled in the state of the art would understand the term schema. By way of example, a database consist of one or more tables, each table has a schema. An XML document may have a schema defining the content. A data management system provides the use of schemas for defining the content of a data set. The organization of the data within the registry can include a multiplicity of schemas. Thus a first data set having a first schema, and a second data set having a second schema, wherein the first data set and the second data set can be logically related to the task at hand.

[0194] An embodiment can use one or more in-core tables to facilitate the registry. Such techniques are known in the state of the art and are provided for with the Daytona Data Management System from Global Technologies Ltd., Inc. See the Daytona User's Guide for details.

[0195] The registry can include encrypted or compressed data and that this is implementation issue. When using the Daytona Data Management System, for example, a record class description can include compressed fields. From the services viewpoint, however, the data is uncompressed until saved by Daytona in a compressed format. Similarly, when the service requests data, the data may be decompressed by Daytona and provided to the service in an uncompressed format.

[0196] The registry can be implemented using horizontal and, or, vertical partitioning techniques. See the Daytona Users Guide for details.

[0197] Administrative functions can be implemented through access methods [access plans] as one would understand the term in database techniques. By way of example, but not limitation, a SQL statement can be used. The implementation, possibly through the use of an ODBC Compliant Driver, (or JDBC Compliant Driver) can create an access plan for accessing and interacting with the data. Similarly, a Daytona Cymbal statement can be compiled into object code and the object contains the access method.

[0198] Administrative functions can be implemented through a 4th generation language such as that of Cymbal, as provided by the Daytona Data Management System [see Daytona's User Guide].

[0199] An embodiment can use one or more components of software to facilitate administering the registry. In such content, the components of software can communicate as required by the embodiment. By way of example, a first component of software on a first computer can communicate with a second component of software executing on a second computer to facilitate an administrative function.

[0200] The schema can be implemented through the techniques of the Daytona Data Management System. The term Record Class Description equates to a schema. A component of software can include the access method for accessing and interacting with the registry.

[0201] The registry can be implemented as a Daytona Project and that one or more administrative functions can be implemented through a first Daytona Application, while additional administrative functions can be implemented through a

second Daytona Application. A Daytona Application has one or more Record Class Definitions. See Daytona's User Guide].

[0202] A registry entry can consist of a multiplicity of information components, and an information component can have an attribute describing the use of the information component. By way of example, but not limitation, an attribute can be PUBLIC, in which case the information component is available to any requesting process. An attribute can be PRIVATE in which case the information component is only accessible to the entity requesting the registration in the registry. An attribute can be SECURE, in which case the information component is accessible to a process satisfying security criteria as defined by the implementation. In the use of attributes, a more robust implementation would define a service associated with the attribute such that the service can be invoked as necessary to perform the functionality desired. By way of example, but not limitation, the PRIVATE attribute can have an associated PRIVATE service that is called by the service accessing the registry, to perform the validation, parsing, filtering, or otherwise data manipulation required to fulfill the functionality of the service. One skilled in the state of the art would understand that such functionality and management of attributed field capability could be implemented with the Daytona Data Management System.

[0203] Program Listings 4.1 through 4.7 provide an embodiment of a directory service. The directory service is started by the generic front end loader, and listens on an Internet Address and port for requests. The directory service reads name/value information components, and acts upon them according to the specified command. The directory service configures the command table during initialization. In the current embodiment, the commands register, create, query, and delete are registered with the directory service. In a second embodiment, additional commands can be registered such as update, modify, replicate, report, and others. In a third embodiment, the commands to be registered can be read from a configuration file,

such as that used by the software engine service. In yet another embodiment, the commands to be registered can be queried from a common directory service. The directory service accesses the request, and locates the command information component. The directory service then locates the corresponding registered command and accesses and interacts with the service associated with that command. By way of example, if the command is register, then the directory service locates the service associated with the register command and accesses and interacts with that service. In the embodiment of Program Listings 4.1 through 4.7, the directory service recognizes the ".private" attribute of an information component and treats such information components accordingly.

[0204] Note that an embodiment of the first directory service can access and interact with a second directory service to determine services to be provided by the first directory service. By way of example, the first directory service can communicate a request for services to the second directory service, and the second directory service can access and interact with the request to determine an appropriate response. The response may include one or more accessible services. This permits a first directory service to be configured according to the criteria supplied by the first directory service to the second directory service. In this regard, the first directory service may have a subset of services that the second directory service supports. By way of example, the first directory service may support a query command, but not a register command. Similarly, the first directory service may support an update command, but not a delete command. By way of example, the first directory service communicates a unique identifier associated with a service provider to the second directory service. The second directory service, responsive to receiving the identifier, accesses and interacts with the registry and determines the unique identifier has a particular security level associated with it. As a result, the second directory service communicates a response indicating one or more commands, and one or more services associated with each command, to the first directory service.

Subsequent use of the first directory service would then be limited to those commands supported by the first directory service.

[0205] A multiplicity of registries can be maintained by the embodiment. Each registry can be accessed by a corresponding directory service. A multiplicity of directory services can be used. Each directory service can broadcast its availability. Such an implementation would use standard broadcasting techniques as defined in UNIX Network Programming series, Second Edition, W. Richard Stevens, Addison Wesley, ISBN 0-13-490012-X, or equivalent thereof. By way of example, a first directory service of a first computer of the network can broadcast its availability. A second directory service of a second computer of the network, responsive to receiving the broadcast from the first directory service, can register the first directory service with the second directory service. Alternatively, the first directory service could access and interact with the second directory service to cause the second directory service to register the first directory service.

[0206] The unique identifier

[0207] The term the unique identifier implies a sequence of characters used to uniquely qualify an entity. In this context, the term entity can represent a consumer, a service provider, a transaction, an entry in a registry, a thread, a process, a function, or a component of software. The reader will be guided by the context of the term to determine the corresponding entity referenced. For example, a consumer the unique identifier is understood as an identifier uniquely qualifying a consumer from other such consumers. A service provider the unique identifier is understood as an identifier uniquely qualifying the service provider from other such service providers.

[0208] The identifier can be a string of characters in the character code set understood by the embodiment. The identifier could contain white space.

[0209] An embodiment can use a multiplicity of strings to ensure uniqueness. For example, an identifier can include a first string and a second string as in:

[0210] IDENTIFIER: Northrup, C., 15 Spring Street, Suite 200, Princeton, NJ

[0211] In this context, the uniqueness may require a multiplicity of information components such as Name, Address, City, State.

[0212] When used in conjunction with a Universal Description Discovery and Interchange Node (UDDI), a uddi_key can be used as the unique identifier.

[0213] When used in conjunction with a hashing service, the registration information, or a portion thereof, provided by the subscriber [ie., the consumer] can be communicated to the hashing service to generate a hash key.

[0214] The unique identifier can include a name value pair, or a multiplicity of name value pairs. This is especially useful when using a directory service to create an entry in the registry. By way of example, a unique identifier can include a first name and value indicating a specific data set (or registry) or logically related data sets. The second name and value pair can indicate a unique key within the data set. By way of example, a unique identifier "sd=payment_services id=cjn@gtlinc.com" would indicate that the service directory (ie. The registry) is called payment_services and id=cjn@gtlinc.com is within that registry.

[0215] A given person may have a multiplicity of the unique identifiers, each the unique identifier uniquely qualifying the person with respect to the activity the person is performing. A person at work may have one the unique identifier for work related activities, a separate identifier for home (or personal) related activities, and a separate identifier for organization activities (such as non-profit organization, little

league, home-school association, political party activities). Note that a person may have the unique identifiers for other activities within an activity.

[0216] A user may interact with a component of software on the user computer to select the current the unique identifier as appropriate for the current activity. The interaction may be through means of a touch screen system, a pointing device such as a Microsoft mouse, speech recognition apparatus, and the like. Regardless of the implementation, software will be used in determining the current the unique identifier. The interaction may cause software to determine the activity and from the activity determine the unique identifier. The aforementioned may be determined solely by a process monitoring the activity of the user, by a process determining the activity of the user, or, by prerecorded information accessible to the process. Furthermore, such process may require interaction or communication with a second process as in the case of a first process communicating with a directory service.

[0217] When the computer system uses the named execution environment of US Patent 5,850,518, then a process can register attributes with the directory service. In such cases, a first user may have access to a first computer, which registers attributes describing a first process on the first computer. The implementation can use this information to deduce or otherwise determine the activity, or, the current the unique identifier, or a combination thereof. When the first user uses a second computer, then a process on the second computer can register attributes with the directory service. In such a case, the first user's activity or the unique identifier, or combination thereof, can be determined by the registered attributes of the second computer.

[0218] When the invention is used with the methods of US Patent 5,850,518, then a first process of the user's computer can communicate with a directory service to determine the current activity or the unique identifier, or combination thereof.

[0219] An implementation can use a unique identifier associated with a user, combined with access and interaction rights based on the network endpoint that the user is connecting from, to determine privilege and authorization. By way of example, a business maintains an enterprise wide network. An employee has an assigned the unique identifier. The employee uses a computer connected directly to the enterprise wide network (i.e., an ethernet behind a firewall). The employee provides their the unique identifier and can access and interact with a service within the enterprise (ie., behind the firewall). The employee leaves the office and goes home. From home, the employee uses an Internet Service Provider, such as America On-Line, to access the Internet. A process on the employee's home computer, connects to the enterprise service executing behind said firewall. The employee provides their the unique identifier. The enterprise service uses an authentication service and determines that the computer the employee is connecting from is outside of the enterprise wide network. The enterprise service then permits the process executing on the employee's computer to access and interact with a limited set of services. The limited set of services may be publically available services that are provided by the enterprise. For example, an administrator within the enterprise may configure the services such that access to customer information will only be granted to a requesting process executing within the enterprise, but, access to the company phone directory is permissible for requesting processes executing outside of the enterprise.

[0220] Dynamically Loadable Module

[0221] A dynamically loadable module is a component of software stored in a shared library, or a dynamic link library, or equivalent thereof, but collectively referred to as shared library throughout this specification. In a typical embodiment, a first function call is made to attach the shared library to the address space of the

requesting process. A second function call is then made to access a particular module within the shared library. It is noted that certain embodiments can take advantage of an initialization function within the shared library that is automatically invoked when the shared library is attached or detached. Examples of this are the DllMain function, or equivalent thereof, as provided by the Microsoft Win32 Interface, and the init function as defined in the KornShell development kit. Various other implementations of shared libraries on UNIX support such initialization functions.

[0222] Function Call and System Call

[0223] For purposes of this disclosure, a function call and a system call are often collectively referred to as a function call. When a particular distinction is necessary, the term system call will be used.

[0224] It must be noted that the AST ToolKit, provided by AT&T Research, and described in Practical Reusable UNIX Software, John Wiley and Sons, ISBN 0-471-05807-6, includes numerous replacement functions via replacement libraries, related to file system access. The replacement functions currently offered by the n-Dimensional File System component of the AST Toolkit and the KornShell, do not augment these standard functions and system calls with access and interactions to services nor to directory service.

[0225] In various embodiments of this invention, a function of a process can be augmented by providing a replacement library containing a replacement function, and using dynamic loading techniques to dynamically load the replacement library (or component thereof), to facilitate the methods and systems of this invention. Alternatively, the corresponding application program could be linked with a library providing functions offering equivalent capability of the replacement function. When reading the term "replacement function" or "augmented function", it is understood as

a function providing an augmented capability or feature which is provided by a replacement function, or a function that the corresponding application program was linked with. Note that this may be in addition to the standard functionality of the corresponding function.

[0226] By way of example, the recv function is frequently used in network programming. (See UNIX Network Programming Volume 1 Second Edition, W. Richard Stevens, Addison Wesley, ISBN 0-13-490012-X.). An embodiment can augment the functionality of the recv function to access and interact with a directory service in order to facilitate administrative functionality such as replication, consistency, communication forwarding, and other services such as wire tapping, broadcasting and the like. Similarly, the functionality could include routing a received request to a second service. Thus, when the process makes the function call, the augmented version of the function can be used to augment or replace the standard functionality of the function.

[0227] By way of example, an augmented function can access and interact with a directory service to determine a service providing the underlying desired functionality. An embodiment could interact with a directory service to determine where the underlying functionality should be executed. A process issuing a write function, for example, could use the replacement write function which would access and interact with the directory service to determine how to access and interact with a write service providing means to write to an accessible device. Similarly, a process issuing a read function call, could use the replacement read function which would access and interact with the directory service to determine how to access and interact with a read service providing means to read from an accessible device. It is understood that such embodiments may require parameter passing from the process issuing the function call, to the service providing the underlying functionality. In such cases, the input types, and possibly the output types may also be communicated

between the process and the service. An implementation could use SOAP/XML for such parameter passing, and possibly for one or more input types, as well as one or more output types. In this manner, a process compiled for a first operating system can be executed on the first operating system, but have one or more augmented function calls accessing and interacting with a service executing on a second computer of the network having a second operating system which may, or may not be different from the first operating system. Note that the service may be a process having means to perform the desired functionality and maintain state.

[0228] A first process can issue an open system call and have a file descriptor (or handle) associated with the opened file, but the file may physically reside on the second computer of the network.

[0229] By way of example, a code fragment written in the C language could include

[0230] int fd=open("/etc/profile",O_RDONLY);

[0231] One skilled in the state of the art would understand that open is a system call and the functionality of the open system call is to open a file identified by the first parameter, which in this case is a file named /etc/profile, for read only. Upon success, the open system call returns a file descriptor value to the process and the file descriptor value is saved in the memory location given by the integer variable field. (For detailed information on the C programming language, see "The C Programming Language, Brian Kernighan and Dennis Ritchie, Prentice Hall Software Series, ISB 0-13-110362-8.)

[0232] When augmenting the open system call, the augmented open function can access and interact with a directory service and specify criteria for selecting a service. By way of example, the criteria could be a service having access to the

/etc/profile file. If such a service is found, then the process can access and interact with the service to cause the service to perform the open system call. The service would have access to the file descriptor associated with the opened file. The service would remain executing, and would provide a response to the requesting process wherein the response indicates a value for the opened file descriptor. The response may be a value indicative of the maximum number of open file descriptors allowed by the operating system, plus the number of opened files that are opened by the service at the request of the process.

[0233] The process can then issue a read function call, and specify the value for the opened file descriptor. The augmented read function would examine the value of the opened file descriptor, and realize it is a value higher than maximum number of opened file descriptors supported by the underlying operating system. In this case, the replacement read function would deduct the maximum number of allowed opened file descriptors from the specified value for the opened file descriptor, and would access and interact with the service providing the underlying read functionality. In this sense, the replacement read function would provide the service with the appropriate file descriptor value, and possible other parameters, and the service would then perform the read system call, and provide the results thereof to the process.

[0234] The communication between the process and the service can be implemented using XML, or using other techniques such as messaging according to a format and possibly a protocol determined by the implementation. In one embodiment, the Safe-Fast-IO (sfio) interfaces are used (See Information Disclosure "Practical Reusable UNIX Software" for details on sfio).

[0235] The process may cause one or more standard functions to be executed on the same computer that the process is executing on. By way of example, certain

environment settings and user administration may need to occur on the same computer as the first process, while other functions can be performed on a second computer according to this invention.

[0236] The process may also require a graphical user interface on the same computer that the process is executing on. In such cases, the functions calls related to the graphical user interface should not be processed by a service executing on a remote computer system.

[0237] The requesting process can register certain function calls that should be executed on the same computer as the requesting process. The augmented function would then determine if the underlying functionality is to be executed on the same computer, or should be executed by the service. To make such determination, the augmented function may access and interact with a directory service having the registered certain functional calls described above.

[0238] Certain functions return a pointer to a memory location. In such cases, an augmented function would access and interact with the service and the service would communicate the results thereof to the process. The communication can include representing data as characters, such as a hexadecimal character or equivalent (such as %32) and the data can be assembled into an allocated memory location accessible to the process. (See communication for details on communication).

[0239] The mapping of one or more return values and side effects of a function performed by a service can be determined by the implementation without changing the scope of the invention. Thus, a service executing a component of software on behalf of a process, can maintain state information about the results of the execution of the component of software, and, can communicate the results and side effects to

the process, which are then assembled and made available to the process as if the function call were completed on the same computer and operating system of the process.

[0240] An embodiment can register additional information components about the devices, services, software, operating system, functionality, communication capability, characteristic and attributes thereof, and other information components as would be necessary to facilitate the invention. By way of example, this can include registration of the service having capability as disclosed herein. Such information may be necessary for the criteria as provided by the process.

[0241] When using name-value pairs, or other identifiers qualifying that portion of a request string which represent a service, the augmented function can use the directory service to discovery the corresponding component of software providing the service. For example, the open function call takes as a parameter, the name of the file to open. However, by providing criteria for accessing the file, the open function call can determine the service it should provide, by interacting with a directory service. By way of example, criteria specified as description="Corporate information about GTL" can be provided to the open function call as the parameter. When the open function calls attempts to open a file with that name, the open will fail. Instead of returning an error condition, the open function call could interact with a directory service to see if there is a service that can satisfy the request. The directory service could either return back entry information and the open function could then access and interact with the service, or, the directory service can connect to the service satisfying the request. Thereafter, when a read function is called, the read function could receive information from the service and provide same to the process. Similarly, when a write function is called, the write function can send the data to the service. Finally, when the close function is called, the close function can disconnect from the service.

[0242] Operating systems typically are deployed with various supporting commands and utilities. By way of example, this often includes a shell, such as ksh. The shell interprets requests and performs desired actions. The POSIX standards define various shell commands and utilities which can be invoked by the shell.

[0243] On a Unix system, such as a Solaris 2.8 operating system, a frequent task is to invoke a cat command to display the content of a file. The cat command takes one or more command line arguments which are file names to display. The output of the cat command is displayed on standard output. Using the shell, one could cat the contents of a file and pipe the standard output as the standard input for a second command.

[0244] The cat command is invoked as a process and the process uses the open function call to open the file. By augmented the open function call with criteria, we can cat the contents provided as a service, as if the content where in a file on the local computer. Thus, the cat command itself does not need to be recompiled to take advantage of this capability. Instead, we use the augmented open function from a dynamically loadable library.

[0245] Similar behavior can be achieved for all standard UNIX commands and utilities that are dynamically linked.

[0246] Similar behavior can be achieved for all standard command and utilities of the U/WIN product line, as well as other applications that are dynamically linked. The U/WIN product line provides the KornShell and the shell commands and utilities for the Windows operating system.

[0247] Registration:

[0248] The registration can be an automated process such that whenever a service begins executing, it always registers its availability with the common directory service. Alternatively, the service can be accessible through a well-defined connection such as a URI, or on a dedicated Internet Address and port. In such cases, the registration process may occur once. In other implementations, a process having appropriate information about the service can register the service. In other implementations, the service may be registered via a user interacting with a graphic rendering program providing a form for the user to complete and submit electronically. Still, in other implementations, the registration process may be via computer mail. The registration process can also be implemented with SOAP/XML techniques. The registration process could also be implemented through remote procedure call, or equivalent thereof. Once skilled in the state of the art would understand that there are a multiplicity of methods for performing the registration process, even calling a person who could manually enter the registration information as required.

[0249] The registration process can include an identifier identifying the directory in which the registration is to occur. By way of example, a registration may indicate: sd="Public Services" in which case the registration is to occur in the Public Services service directory. A default directory can be used when the registration process does not provide such a service directory identifier.

[0250] The registration information includes information provided by the process (or processes) participating in the registration. The registration information is said to contain one or more information components. An implementation can use a name-value pair for an information component, such as name="Charles Northrup", or, can use XML to communicate the information component, or various other techniques which may, or may not require a schema.

[0251] The implementation can support private and public attributes, as described in US Patent 5,850,518. In such cases, an information component can be marked as private, and thus would be accessible only to the directory service, but would not be returned in queries or reports. A private information component is always accessible to the administrator of the directory service. Similarly, a private information component is always accessible to the owner of the service.

[0252] An information component can be marked with a Group attribute. According, members of the specified group (or processes acting on their behalf) would have access to the information component.

[0253] Implementations can use underlying operating system security semantics as well. For example, a Unix system supports the notion of read/write/execute permissions for owner, group, and others. Such operating system semantics can be used.

[0254] The registration process can include the use of a graphical interface to make the registration experience more pleasurable for the user. Such implementations could be facilitated through the use of the Microsoft Internet Explorer or equivalent thereof. Alternatively, the graphical interface can be provided by other means, as one skilled in the state of the art would understand.

[0255] Note that some implementations will have the directory service provide required registration information to the registering process, and that such information may be communicated to a user of a computer system, and that the user would provide the required information and the required information would then be accessible to the directory service.

[0256] The registration information is administered by the directory service, which can use a registry to provide persistence for the data.

[0257] A service provider can register a multiplicity of registrations with the common directory service. This can permit artificial intelligence methods for the selection of the service satisfying criteria. The selection can include events, time specifications, access methods, communication methods, methods providing selection based on response times, and the like. In such cases, a service provider can register that the service provided by the service provider at a particular network endpoint is accessible only during certain hours of operations, which may include day of week, month, year, etc. The same service can be registered for a different network endpoint for a different hour of operations, which may include day of week, month, year, etc. The only restriction is that duplicate entries in a single service directory are not supported.

[0258] It is noted that replication of entries between service directories registries may be provided by the implementation. In such cases, a first directory service can provide a second directory service with one or more registration entries maintained by the first directory service, in order to replicate the data maintained in the registry. An implementation can use the methods of US Patent 5,572,709, or equivalent thereof. Each time an entry is written to the registry, the write(2) system call can be augmented to duplicate the write request to a remote file store. The write(2) system call can also use the directory service to determine a remote process having capability to receive registry updates. The write(2) system call can connect to the remote process and communicate the information related to the write system call. The remote process would receive the communication and perform equivalent action to a data store maintained by the remote process. The remote process can either update its registry immediately, or, store the communication until sufficient communications have been received, and use bulk data loading techniques to bulk load the data.

[0259] In a second implementation, a first directory service receives requests, and depending on the request received, will duplicate the request to a second directory service. By way of example, the first directory service receives a request. The request is scanned to determine if the request is to register a new service, and if so, the first directory service accesses and interacts with a remote directory service to replicate the request. This would be in addition to the first directory service performing the operations of the received request.

[0260] To maintain consistency, other request such as delete, modify, change, update, and others can also be replicated.

[0261] The implementation can provide this capability in a function of a dynamically-linkable replacement library. One example of a dynamically-linkable replacement library is found in US Patent 5,572,709.

[0262] By way of example, a gethostbyname standard operating system interface call can be augmented to access and interact with a directory service as required. (See UNIX Network Programming Networking APIs, UNIX Network Programming series, Second Edition, by W. Richard Stevens, pp 240-246, ISBN 0-13-490012-X for details on the standard gethostbyname operating system interface.) Program Listing 1.1 provides a source code listing of one implementation for the replacement gethostbyname function which is then compiled into object code, and archived in a replacement shared library with the same filename as the standard shared library containing the operating system provided gethostbyname function. Using the LD_LIBRARY_PATH environment variable setting to first point to the location of the replacement shared library; the replacement gethostbyname function would be used whenever a process invokes the gethostbyname function.

[0263] The standard system version of the gethostbyname function accepts a single parameter hostname, which is a pointer to a character string and returns a

pointer to a hostent structure on success, or a NULL pointer on failure (Program Listing 1.1 line 3).

[0264] In this embodiment, the gethostbyname function will first invoke the system version of the gethostbyname function (Program Listing 1.1 line 8) to see if it is able to resolve the host name reference given by the value pointed to by parameter hostname.

[0265] If the system version of the gethostbyname function is not able to resolve the hostname, then the gethostbyname function will consider the host name reference given by the value pointed to by parameter hostname as criteria for selecting a service. In this case the gethostbyname function will query the directory service (Program Listing 1.1 line 12) and will examine the results of that function to see if connectivity has been registered for a service satisfying the criteria (Program Listing 1.1 lines 13-18). In this case, the gethostbyname function will then invoke the system version of the gethostbyname function (Program Listing 1.1 line 19).

[0266] In a second embodiment, the standard operating system interface call can include the necessary computer instructions to access and interact with the directory service.

[0267] Other embodiments are possible. By way of example, the gethostbyaddr, gethostbyname2, getservbyname, getservbyport, getnameinfo, and others, can have appropriate replacement functions to access and interact with the directory service. This is not limited to socket application programming interfaces. By way of example, an open system call can be modified to access and interact with a service, through the use of a directory service.

[0268] The benefit of using replacement dynamically loadable libraries is that the original source code for the application program need not be modified to gain the advantage of working with the directory service. Thus, applications, such as telnet, Netscape communicator, ftp, ping, and others, can immediately take advantage of the directory service, without having to recompile the application.

[0269] By using a replacement dynamic link library with an alternative gethostbyname function, the user can enter information that can then be communicated to a directory service, and the appropriate response displayed.

[0270] In an enterprise network, such as within the Global Technologies Ltd., Inc., domain (gtlinc.com), we can maintain a registry containing contact information for our employees. When using the browser, a first employee can enter "contact information for Charles Northrup" and the directory service locates a service providing that information, accesses and interacts with the service, and communicates the response from the service, to the browser process.

[0271] Netscape 4.73 and Microsoft Internet Explorer version 5.0 permit the user to enter a string. Both products attempt to resolve the entry by using a domain name lookup service, usually provided by gethostbyname (or equivalent thereof). When a domain name cannot be determined, both products will interact with web search engines to determine a relevant page. By way of example, the Microsoft Internet Explorer will communicate with the Microsoft Service Network search engine site. If the string was entered as C:\, then both products insert a file schema and as translate the request as file:///C|/. Neither product permits access and interaction with a directory service.

[0272] The implementation can also be provided directly by the operating system interfaces themselves.

[0273] An example directory service is shown in Program Listings 4.1 through 4.7. The embodiment provides for a register command, a create command, a query command, and a delete command. When registration is to occur, the name/value pair may include a ".private" notation to indicate that the name/value pair is private, and should not be reported as part of a query command. As an example:

[0274] Name="charles northrup" phone.private=609-924-7305

[0275] In this context, the registration entry will include two information components. The first is a name component, having value "charles northrup" and the second is a phone component having value 609-924-7305. When querying the directory service using:

[0276] Command=query name="charles northrup"

[0277] then the query will report the name component and its value, but not report the phone component nor its value.

[0278] An implementation can add a ".mandatory" attribute to an information component to force the specified information component to be included in a query request. By way of example,

[0279] Command=register name="charles northrup" phone.mandatory=609-924-7305

[0280] In this example, a query request must include phone=609-924-7305 in order for the entry to be included in the query results.

[0281] An implementation can add a ".group" attribute to an information component such that the a group list is maintained by the directory service, and only those belonging to the group can see the results of the query. By way of example:

[0282] Command=register name="charles northrup" group.mandatory=officer

[0283] In this example, a query request with criteria name="charles northrup" would require the request process to supply additional information so that the directory service can determine if the request is on behalf of a member of the specified group.

[0284] Note that the use of the attributes can be extended to a connect request facilitated by the directory service. In such cases, a request of:

[0285] Command=connect name="charles northrup"

[0286] Would be subject to the same constraints as the query command, as described above.

[0287] In assigning attributes to information components within a registry entry, an implementation can use the directory service itself to access and interact with a service providing the desired functionality. By way of example, the private attribute described above can be a registered service within the common directory service (CDS). When the CDS receives a query command, and locates one or more entries satisfying the request, the CDS could access and interact with a "private" service which could perform translation to an empty string for that information component. In a another implementation, an information component can have a "normalize_to_upper" attribute and the CDS would access and interact with the

service providing normalize_to_upper normalization of the data content for the value portion of the name / value information component.

[0288] Registration Information

[0289] By way of example, but not limitation, this may include one or more of:

- consumer information
- name
- street address
- city
- state
- country
- postal code
- information representative of the consumer computer
- information representative of the operating system of the consumer computer
- information representative of the communication devices of the consumer computer
- information representative of components of software accessible to the consumer computer
- consumer contact information such:
- telephone number
- fax number
- beeper number
- pager number
- wireless access number
- cellular phone number
- company information
- affiliation information

- corporation information
- non-profit business information
- organization information
- agency information
- consumer add-on services
- consumer subscribed services
- consumer billing information
- consumer payment information
- consumer historical usage information
- consumer historical payment information
- consumer transaction information
- consumer security information
- consumer profile information
- consumer access information
- consumer geographical information
- consumer preference information
- consumer enhancement service information
- payment type
- payment provider unique id
- payment account unique id
- payment billing information
- payment billing name
- payment authorization unique id
- payment provider id assigned expiration date
- payment provider code
- payment bank unique id
- payment bank authorization unique
- connectivity required to reach a service
- access point

- Internet address
- port
- protocol
- network type
- data representation
- service availability time
- duration of service
- owner information
- group information

[0290] When used in conjunction with the methods of US Patent 5,850,518, the information can include one or more of the information components as defined in the thread directory service. By way of example, but not limitation, this can include the physical connectivity required to reach the consumer, the consumer service, or any service including a minor service, a communication primitive to be used in communications wherein the information on the physical connectivity required is used by the communication primitive to establish connectivity. As an example, a consumer computing device connects to the Internet through an Internet Service Provider [ISP] and is assigned a dynamic Internet Address. The registration information can include the dynamic Internet address and possibly one a port for sending and receiving communications. One skilled in the state of the art would understand that a multiplicity of ports may be used in facilitating the communication.

[0291] Alternatively, if the consumer computing device has a static Internet Address associated with it, that the static Internet Address and a designated port can be registered. One skilled in the state of the art would understand that a network address and possibly a port number, or equivalent thereof, can be used. By way of example, an Internet Address may be 192.127.0.3 and a port may be 3999.

Alternatively, an Internet Address can be workhorse.gtlinc.com and a binding service such as that provided by the name daemon or equivalent thereof would bind workhorse.gtlinc.com to an appropriate network address.

[0292] Accesses and Interacts

[0293] The phrase accesses and interacts implies the use of a multiplicity of processes. The processes may communicate via interprocess communications, intraprocess communication, or through a communication device as supported by the underlying operating system. Communications can be instrumented through protocols. A first process can be executing on a first computer of the network, and a second process executing on a second computer of the network. It is understood that this may include one or more intermediary processes to facilitate the communication, as determined by the protocol. It is understood this may include one or more intermediary processes to facilitate the communication, as determined by the network. The network can be the Internet, a private network, a public network, or some other network such as the virtual network as described in US Patent 5,850,518.

[0294] The phrase access(es) and interacts can also imply loading a dynamically loadable module into the address space of the first process and invoking a function entry point in the dynamically loadable module either directory, or indirectly through an initialization function supported by the underlying implementation. By way of example, the DllMain function is invoked whenever a dynamically linked library is attached to a process.

[0295] Criteria

[0296] Criteria can be implemented through name/value pairs, which may include using regular expressions and possibly even using Boolean operators, through

SQL statements, through OBDC instructions, JDBC instructions, Microsoft ADO.NET, through Daytona Cymbal statements, and other implementations. The interpretation of the specification of the criteria is implementation dependent. Various protocols can also be used. A natural language system could be used in conjunction with the directory service, to interpret the criteria. Examples of Natural Language Systems include CHAT, from Network Services and Interfaces Laboratory, Communications Research Centre, 3701 Carling Ave. Ottawa, ON CANADA K2H 8S2. Additional technical papers include A Form-Based Natural Language Front-End to a CIM Database, Nabil R. Adam, Aryya Gangopadhyay, March-April 1997 (Vol. 9, No. 2), Knowledge and Data Engineering, IEEE (also available at http://www.computer.org/tkde/tk1997/k0238abs.htm).

[0297] Preprocess

[0298] The term preprocess, as used in this specification, indicates a service that is to be performed on a communication prior to primary processing. By way of example, this may result in a second memory location being made available to the process wherein the second memory location has the results of the preprocessing. By way of example, the preprocess service may:

- translate a communication
- interact with a service to alter the communication such as macro expansion, or regular expression evaluation
- decrypt the communication
- unscramble the communication
- access and interact with a directory service to ascertain information relevant to the communication
- convert a component of the communication from a first format to a second format, such as converting a string to a hexadecimal number, an integer, a binary value, ..etc

- convert a component of the communication from a first arbitrary named representation to a second arbitrary named representation
- normalize a component of the communication, such as in changing the case, the format, or the data representation.

[0299] The preprocess service may be dynamically loadable. The implementation may determine which preprocess service to dynamically load. Such determination could be made by accessing and interacting with a directory service, and possibly by using a component of the communication.

[0300] By way of example, a first process receives a communication and examines the communication for a name/value pair. The first process uses the name/value pair as criteria for selecting a preprocess service. The first process accesses and interacts with the preprocess service.

[0301] A communication received by the first process can be encrypted according to a first encryption method. The first process would then access and interact with a service providing decryption of the communication according to the first encryption method. The same first process can receive a second communication encrypted according to a second encryption method. The first process would then access and interact with a service providing decryption of the communication according to the second encryption method. By selectively accessing and interacting with the preprocess service, additional encryption / decryption methods can be devised in the future and the first process will be able to take advantage of same without having to recompile the first process.

[0302] A communication received by the first process can be formatted according to a first protocol. The first process can access and interact with a service providing translation of the communication from the first protocol to a second

protocol. The first process would then process the communication according to the second protocol.

[0303] A communication received by the first process can be formatted according to a first language. The first process can access and interact with a service providing translation of the communication from the first language to a second language. The first process would then process the communication according to the second language definition.

[0304] A communication received by the first process includes a mixture of upper case and lower case characters. The first process can access and interact with a normalization service providing means to convert one or more of the lower case characters to upper case, or upper case to lower case as determined by the implementation. By way of example, the URL http://www.gtlinc.com/research/research.html, can have a portion of the URL normalized, while the remainder of the URL remains as received. One implementation can convert http://www.gtlinc.com, from lower case to upper case, while a second implementation may convert from upper case to lower case. When the communication includes a component which is relative to a well known root, then the normalization may convert the relative portion to a fully expanded name which includes the root. By way of example, a relative URL given as research/research.html, may be normalized to the fully qualified name of http://www.gtlinc.com/research/research.html.

[0305] Note that it is possible for a NULL preprocess service to be indicated to the first process, in which case, the first process would not call the preprocess function.

[0306] Note that a preprocess service may allocate and initialize even in part, a memory location to be used by the first process.

[0307] Postprocess

[0308] A first process may access and interact with a post processing service. In the case of a software engine, the postprocess service performs deallocation and garbage collection of memory allocated, frequently by the preprocess service. Postprocessing can also include translation, formatting, normalization, and even encryption of a response, prior to sending the response.

[0309] Common Directory Service

[0310] The phrase common directory service implies a directory service accessible to a requesting process (or a service), and, containing information related to a desired service. A component of software can be used on a first computer of the network to communicate with the directory service executing on a second computer of the network. An implementation can use a multiplicity of directory services, and, that a directory service may access and interact with additional directory services, as necessary. A process may also be configured to have direct access to the directory service as a function of the process. In such cases, the process invokes a function providing the administrative feature desired (i.e., registration, query, unregister, modify, update, create, join, select, ..etc).

[0311] Facilitates the Connection On Behalf of the Requesting Process

[0312] The phrase facilitates the connection on behalf of the requesting process implies the directory service connects the requesting process to the desired service. One skilled in the state of the art would understand that an implementation of the

directory service could provide the required connectivity to reach the service, to the requesting process, and the requesting process could connect to the service. One skilled in the state of the art would also understand that an implementation of the directory service could include the Thread Communication Service as disclosed in US Patent 5,850,518. One skilled in the state of the art would also understand that there are variations of the implementation within the scope of the invention that can be used to facilitate the connection.

DETAILED DESCRIPTION

[0313] A service is executing on a computer system on a network. The service can be listening on a network endpoint, such as an Internet address and port. The implementation can use the socket application programming interface, or some other method as provided by the underlying operating system interfaces for communication connectivity and synchronization. For the service to be used by a requesting process, the service must first be registered (see registration) with a common directory service.

[0314] The requesting process begins execution, and accesses and interacts with a common directory service. The requesting process specifies criteria for a desired service.

[0315] The common directory service locates a service entry satisfying the criteria, and facilitates the connection on behalf of the requesting process to the desired service.

[0316] A user of the computer may be interacting with the requesting process. The user can cause the computer operating system to access and interact with a process to complete registration.

[0317] The user can communicate a request for a service to a requesting process. In this context, the user is referred to as a consumer. The requesting process would then access and interact with the common directory service on behalf of the user. The requesting process, which may first need to preprocess the user request, can provide the request to the directory service.

[0318] The requesting process can then access and interact with the desired service.

[0319] Since the user is a registered user, the service can access and interact with the common directory to determine public registration information components about the user.

[0320] Similarly, the requesting process can access and interact with the common directory service to determine public registration information about the service. In certain implementations, the requesting process can access and interact with the common directory service to determine public registration information about the provider of the service. The requesting process may communicate such determined information to the user, either through audio or graphically through the use of a graphical user interface, or text based as one might use the curses library available on UNIX derived implementations. The requesting process may access and interact with a component of software accessible to the requesting process to filter out certain services deemed inappropriate or undesirable.

[0321] A registered consumer can also provide a service. To provide the service, the consumer must register the service with the common directory service. Once registration is complete, the service provided by the consumer will be accessible through the common directory service.

[0322] A service provided by the consumer can be implemented with a callback capability. By way of example, a consumer request a service from a service provider, and the consumer must pay for the service. In this context, the consumer supplies service provider with access to the consumer's the unique identifier. The service provider service accesses and interacts with the common directory service to request access to the consumer's payment information service. The common directory service locates the consumer service satisfying the request, and creates a transaction identifier to indicate that a transaction is in progress. The common directory service can complete a registration of the pending transaction. The common directory service then accesses and interacts with the consumer payment service. It provides the pending transaction unique identifier to the consumer payment service, and then disconnects. The consumer payment service then accesses and interacts with the common service directory specifying criteria including the unique identifier of the pending transaction registration entry. In this context, the consumer payment service is now a requesting process. The common directory service then facilitates the connection on behalf of the requesting process to the service provider process awaiting payment information, as identified by corresponding transaction id.

[0323] When using the invention on a computer system behind a firewall, a consumer providing a service may request a service provider providing a service hosting service, to host the consumer service on behalf of the consumer.

[0324] When using the invention on a computer system behind a firewall, the service can complete registration with the common directory service indicating that request to access the service from the common directory service are to be facilitated through a protocol, such as computer mail protocol. Thus, the common directory service would send computer mail to the owner of the service, and a service process executing on the computer system would read the computer mail and determine that

there is a request for the service. The service process would then facilitate a connection to the central directory service. In one embodiment, the service process can provide the request to the requested service and the requested service would then access and interact with the common directory service. In a second embodiment, the service process would access and interact with the common directory service on behalf of the requested service. It is noted that an implementation can use various methods for automating the registration process in this regard, and defaulting to computer mail protocol when other types of connectivity cannot be established.

[0325] When accessing and interacting with a service, a requesting process can communicate according to a first protocol, which is then brokered by the common directory service to a second protocol as required by the service. Alternatively, a service process can communicate according to a first protocol, which is then brokered by the common directory service to a second protocol as required by the requesting process. In such cases, this may include the use of language translations. By way of example, a first language can be formatted according to the rules of a second language. The broker service can use a translation service to perform such translations according to well-defined rules. The translation service may also use templates as required.

[0326] Various embodiments of using services, communication flow between services, registration of services, ordering of registration, ordering of callbacks, are presented. Once skilled in the state of the art would understand that various permutations are permitted by the invention. Thus, a callback in one service could easily be implemented in a second service, as appropriate.

[0327] Additional Services

[0328] Numerous additional services can be added to the consumer service, the provider service, or the central service. Such additional services are contemplative of means to facilitate the transaction, and to ease the burden of administering the data associated with the transaction. Examples of such services are:

- a component of software accessible to the consumer computer which activates upon notification of pending transaction, to alert the consumer that a transaction is now in progress. The alert may be audio or visual, or combination thereof. By way of example, the component of software may cause an icon to become visible while the transaction is in progress.
- a component of software accessible to the consumer computer that activates upon notification of pending transaction and requires the consumer to select an "Authorize Payment Information Transfer" option. If the consumer does not select the option within a predefined period of time, then the transaction would abort. Alternatively, a "Decline Payment Information Transfer" option may be selected by the consumer, and the transaction would abort. By way of example, the component of software may display such options as graphical representations for the user to select via depressing a mouse button (i.e., "click").
- a component of software accessible to the consumer computer to permit the consumer to select a current payment option from a plurality of payments options available to the consumer. By way of example, a consumer has a multiplicity of credit cards and maintains payment information for each such credit card. When notified of a pending transaction, the consumer can select which of the registered credit cards is to be used.
- a component of software accessible to the consumer computer to permit the consumer to select a current payment option from a plurality of payment options communicated by the service provider to the central

service, and from the central service to the consumer service. By way of example, a service provider may accept only American Express credit cards. By communicating this to the central service, and from the central service to the consumer service, then the consumer service can alert the consumer that American Express is the only credit card accepted by the service provider. Accordingly, the consumer service can automatically select the American Express registered information and communicate such information back to the central service [and from the central service to the service provider].

- a component of software accessible to the consumer computer to temporarily deactivate, or to terminate, the consumer service providing the payment information. By way of example, the consumer can use a mouse connected to the computing device to select an icon, such as a wallet, or a purse, to indicate that it is to be closed. In closing the wallet, the consumer service providing the payment information would then be deactivated or terminated. Similarly, selecting the same icon [or a different icon which graphically conveys the notion that the wallet is closed], can cause the consumer service to reactive. In such instances one can use a first icon to indicate the consumer service is not executing, and a second icon to indicate the consumer service is executing.
- a component of software accessible to the consumer, providing means to communicate with the central service. This provides means interact with the central service to facilitate transaction reports, to inquiry on service providers having registry entries containing the consumer the unique identifier, to change keywords recorded in the registry for the consumer for a specified service provider, or, for facilitating administrative functionality as one would anticipate for a consumer / service provider relationship.

- a component of software accessible to the central service to alert consumer of transaction in progress. This may include changing an icon from a first color to a second color to indicate the transaction in progress, and back to the first color once the transaction is complete. Alternatively, the component of software can display one of two different icons to indicate the current state as either in progress, or, transaction complete.
- a component of software accessible to the central service to periodically verify the consumer through interaction with the consumer service.
- a component of software accessible to the central used by the central to track frequency of use based on statistical analysis to alert for possible fraud.
- the service provider can communicate payment options to the central. The central can send the information to the consumer service. The consumer service can alert the consumer to the available payment options, and the consumer can select a particular option for that transaction.
- the service provider can communicate total payment required to the central. The central can send the information to the consumer service. The consumer service can alert the user and possibly request authorization based on the total amount of the payment.
- the service provider can communicate transaction details to the central, which then communicates the information to the consumer service, providing the means for the consumer service to detail the information on the consumer computer.
- a component of software accessible to the service provider process which communicates with the central service, to provide certain consumer registry information to the service provider. In this context, a field marked as PUBLIC, can be returned to the service provider. Thus, a

consumer can supply the service provider with the consumer's the unique identifier, and the service provider can contact the central service, communicate the consumer's the unique identifier, and receive a communication from the central service representative of the consumer's registered information that is publicly available through the registry implementation.

[0329] Software Engine Service

[0330] A service can be instrumented through a software engine. The software engine uses a specification describing one or more components of the engine. The specification is referred to as the engine configuration specification. The components of the engine are referred to as the component services. Note that a component service provides a service, and hence the component service is often referred to, more simply, as service. An example of an engine configuration is provided in Program Listing 2.16.

[0331] A minimal engine specification contains sufficient information for the software engine to associate the component identified in the specification with a service, which may be provided by a component of software. As such, the engine can access and interact with the service as necessary to perform the desired action.

[0332] As an example, a specification can identify a given service that is to be dynamically loaded through the use of one or more operating system interfaces.

[0333] It is expressly understood that the specification of the engine components can be facilitated through a schema. In use with the Daytona Data Management System, a record class provides equivalence of a schema.

[0334] Similarly, a specification can be facilitated through the use of one or more data structures.

[0335] Similarly, a specification can be facilitated through the use of one or more name spaces. A name space may be facilitated by the operating system, an application having means to interpret a name space, a middleware layer having means to interpret the name space, an interpretive language processor having means to interpret the name space, or through the use of a directory service such as LDAP, Microsoft Active Directory, or the Thread Directory Service of US Patent 5,850,518. By way of example, but not limitation, a name space could be given as:

[0336] engine=

[0337] (

[0338] component=authenticate

[0339] (

[0340] name=auth;

[0341] location=libservices.so.1.0;

[0342])

[0343] component=input

[0344] (

[0345] name=readline;

[0346] location=libservices.so.1.0;

[0347] physical=127.0.0.1:9998

[0348])

[0349])

[0350] A similar specification could have been given as:

[0351] engine=test_engine

- engine.authenticate.name=auth;
- engine.authenticate.location=libservices.so.1.0;
- engine.input.name=readline
- engine.input.primitive=inet
- engine.input.physical=127.0.0.1:9998

[0352] Various methods for providing the specification could be implemented through various name space techniques. Such techniques could include the use of SOAP/XML, XML, or other protocol and, or language specifications.

[0353] By way of example, but not limitation, the engine could be designed to:

- communicate with a service to discover the specification, or
- communicate with a service that sends the specification to the engine, or
- access and interact with an accessible file to determine the specification, or
- access and interact with environment settings to determine the specification, or
- access and interact with operating system interfaces to determine the specification., or
- access and interact with a service to discover the specification, or
- use one or more of the above to determine the specification.

[0354] When an engine must determine the data type of a specification component, the engine could access and interact with a service providing such information. By way of example, but not limitation, ODBC, JDBC, backtalk, XML schemas, and other such methods can be used. One skilled in the state of the art should interpret this to imply that the engine can interact with a service providing the detailed information on one or more components of the specification in order to determine the data type.

[0355] Alternatively, the engine can use a binding service such as that provided for in US Patent 5,850.518 to determine the association of an identifier with an entity understood by the binding service. By way of example, but not limitation, the binding service can use a method providing means to associate the identifier to a data

type. The engine can then request information from the binding service to receive the data type information. In such cases, the binding service method can use a service such as ODBC, JDBC, backtalk, XML schemas, or other such methods as appropriate.

[0356] An engine can be implemented with a services of components preconfigured, but dynamically loaded as specified by the specification. By way of example, a standard engine could provide:

- authenticate - a service for determining if the requesting process is authorized to use the service
- input - a service providing means to receive input
- preprocess - a service providing preprocessing of the input
- process - a service providing primary processing
- postprocess - a service providing postprocessing of a response
- response - a service providing a response

[0357] The standard engine can also access and interact with one or more of: a startup service, a shutdown, and an engine configuration service.

[0358] A specification for the standard engine may include:

[0359] Component=authenticate name=auth location=libauthenticate.so.1.0

[0360] This would instruct the engine to dynamically load the authenticate service given in the dynamically loadable library called libauthenticate.so.1.0, and module name auth.

[0361] When configured with a directory service, the above specification could be given as:

[0362] Component=authenticate name=auth

[0363] This would instruct the engine to use a directory service to locate the service named auth, and to access and interact with the service accordingly.

[0364] A specification for a standard engine may include a placeholder service for a component. In this case, the engine will access and interact with the placeholder service even though the placeholder service contains a simple return statement or exit statement and performs no other action.

[0365] An authentication service can be implemented to determine if the process accessing and interacting with the service, is permitted access to the full capability of the engine. For example, an authentication configuration file can store information indicating a host, and indicating if the service is allowed or denied according to the host. The authentication service can then access and interact with the authentication configuration file to determine is full access is granted.

[0366] Authentication can include receiving a unique identifier assigned to an entity providing a service (or a registered user), and determining if the entity is permitted according to the rules of the authentication service. By way of example, an authentication configuration file can include:

[0367] CID=0x1924865319279337 host=gtlinc.com command=allow

[0368] CID=0x1924865319279337 host=* command=deny

[0369] When the authentication service is invoked, the host computer requesting the service must be gtlinc.com and the request must include the CID value 0x1924865319279337.

[0370] The authentication service configuration specification can require the authentication service to access and interact with a directory service that a specified component appears in a registry entry. By way of example, a configuration specification of:

[0371] criteria="host=gtlinc.com cid=?" command=allow

[0372] would cause the authentication service to fill in the cid value according to the received communication, and provide that name value pair, as well as host=gtlinc.com name / value pair, as criteria for the directory service to determine if the specified cid entry contains host=gtlinc.com. If so, the engine would continue processing, otherwise, the engine would deny access. Note that in this example configuration specification, the value of cid=? would be interpreted by the service as a macro expansion to be completed by the service. In this case, the service can use a component of the communication, or equivalent thereof, to complete the value portion of the name/value pair.

[0373] It is understood that when the authentication service must receive a communication containing an information component, then the authentication service may access and interact with an input service and possibly a preprocess service, before authentication can be completed. This may be necessary when the authentication service requires the requesting process to provide name/value pairs.

[0374] Note that unlike the UNIX inetd process, which can use a TCP Wrappers implementation to determine if the request from a remote system is authenticated, the use of the engine is on a per engine basis. Each authentication service can have its own authentication configuration specification, regardless of the network endpoint on which the requesting process is listening on. Similarly, each authentication service can have its own authentication configuration required for a

two-way handshake when a requesting process connects to a service. That is to say, the requesting process can use the authentication service to verify the connected service, just as easily as the connected service using an authentication service to verify the requesting process has access rights to the service.

[0375] In the embodiment provided in Program Listing 2.0 through 2.18, the software engine is configured to access and interact with a startup service, if defined in the engine configuration. Similarly, if the engine configuration specifies a shutdown service, then the engine uses the atexit operating system interface to cause the shutdown service to be invoked when the engine terminates. The basic engine components are given as authorize, input, preprocess, process, postprocess, and response. Placeholder services are used for each of the aforementioned services. During initialization, the engine accesses and interacts with the configuration to determine what service components are specified, and how to access and interact with them.

[0376] The startup service is typically used for memory allocation of one or more data structures used by the components. In general terms, the startup service performs resource initialization. By way of example, the startup service may access and interact with the common directory service to determine available services, entities providing services, characteristics of entities or services, registration, and similar operations.

[0377] The shutdown service is typically used for memory deallocation and performing closure routines. The shutdown service, in general terms, deallocates resources. By way of example, the shutdown service may access and interact with the common directory service to deallocate resources, deregister, or perform other operations. On a Windows operating system, the embodiment may use the atexit function, or equivalent thereof.

[0378] Program Listing 2.17 shows a second embodiment of the software engine. In this embodiment, each time the engine is called, the engine will call the configure_engine service (function) to perform engine configuration. In this sense, each time the engine is to do something, it will always reread the configuration specification to determine the current engine components. This permits a first set of engine components to be provided in a first engine configuration specification, and a second set of engine components to be used for subsequent engine processing.

[0379] When used with the generic front end loading service (gfel), we can specify that the engine provides a service by invoking gfel with the appropriate parameters. By way of example:

[0380] gfel name=engine location=libengine.so.1.0 primitive=INET physical=192.168.200.15:999

[0381] causes gfel to start the engine listening at internet address 192.168.200.15 port 999. When used with gfel, the engine can include an administrative service such that when accessing and interacting the administrative service, the engine component parts can be reordered, replaced, or otherwise permitting dynamic reconfiguration of the engine.

[0382] Program Listing 2.18 shows a third embodiment of the software engine. In this embodiment, the number of engine components parts, nor their ordering, are predetermined by the engine. Instead, the components are determined by reading an engine component specification. Component ordering is maintained based on fifo ordering. In an alternative embodiment, a hash list, or other mechanisms known in the state of the art can be used. By way of example, component ordering can be determined by specifying the component order number in the configuration specification, or, by deducing component order specification based on dependency, or, establishing component ordering based on rules. The engine can determine the

components and their ordering by processing the engine configuration specification. In an alternative embodiment, the engine could access and interact with a service to determine the engine component specification. For example, the engine could access and interact with a common directory service to query for information components containing a keyword such as keyword=engine.conf, and use the results of the query to configure the engine.

[0383] Authentication Service

[0384] An authentication service provides authentication for use of a service. A widely used and well known authentication mechanism is tcpwrappers.

[0385] The following paragraphs are from The Red Hat Linux 7.2: The Official Red Hat Linux Reference Guide:

[0386] TCP wrappers and xinetd control access to services by hostname and IP addresses. In addition, these tools also include logging and utilization management capabilities that are easy to configure.

[0387] Many modern network services, such as SSH, Telnet, and FTP, make use of TCP wrappers, a program that is designed to stand between an incoming request and the requested service.

[0388] The idea behind TCP wrappers is that, rather than allowing an incoming client connection to communicate directly with a network service daemon running as a separate process on a server system, the target of the request is "wrapped" by another program, allowing a greater degree of access control and logging of who is attempting to use the service.

[0389] The functionality behind TCP wrappers is provided by libwrap.a, a library that network services, such as xinetd, sshd, and portmap, are compiled against. Additional network services, even networking programs you may write, can be compiled again libwrap.a to provide this functionality. Red Hat Linux bundles the necessary TCP wrapper programs and library in the tcp_wrappers-<version> RPM file.

[0390] When someone attempts to access a network service using TCP wrappers, a small wrapper program reports the name of the service requested and the client's host information. The wrapper program does not directly send any information back to the

[0391] client, and after the access control directives are satisfied, the wrapper gets out of the way, not placing any additional overhead on the communication between the client and server.

[0392] TCP wrappers provide two basic advantages over other network service control techniques:

[0393] The connecting client is unaware that TCP wrappers are in use. Legitimate users will not notice anything different, and attackers never receive any additional information about why their attempted connections failed.

[0394] TCP wrappers operate in a manner that is separate from the applications the wrapper program protects. This allows many applications to share a common set of configuration files for simpler management.

[0395] Thus, an application program must be linked with the libwrap.a library. Once deployed to the field (i.e. a customer site), then the application program is static with well defined functionality. Thus, a replacement tcpwrapper cannot be used,

unless the application program is recompiled (i.e., linked against libwrap.a) and redeployed.

[0396] Another disadvantage is that tcpwrappers can be used to authenticate a request for a particular application program from a client at given Internet Address, but does not authenticate individual services provided by the application program. A given application process can use tcpwrappers to authenticate based for the primary service provided by the application process, but, does not use tcpwrappers to authenticate for minor services provided by the application process.

[0397] By extending the capability to minor services offered by a primary service, we can provide a greater level of authentication and access control.

[0398] By way of example, an authentication service embodiment is provided in Program Listing 3.0. The authentication service is used by the engine service. Thus, we can use tcpwrappers to authenticate for the engine service, and use our own authentication service within the engine, based on the engine component specification. That is to say, when the engine service is configured, we can include the authentication service as a component of the engine. This permits authentication using client Internet Address to determine accessibility to one or more minor services provided by the application service. Alternatively, we could use the domain name associated with the requesting process.

[0399] By way of example, an engine component providing input to the engine, can access and interact with the authentication service to determine if the requesting process has appropriate authorization to use the service provided by the input component.

[0400] By way of example, an engine component providing preprocessing of input, can access and interact with the authentication service to determine if the requesting process has appropriate authorization to use the service provided by the preprocessing component.

[0401] By way of example, an engine component can access and interact with the authentication service to determine an appropriate replacement component for the engine based on the client credentials, which could include the Internet address, the domain name, or other information such as a variable name and value. By way of examples, an information could be "name=c.northrup." Various variable naming techniques, such as that provided by the KornShell command and programming language could be used. As another example, when a requesting process is executing on a computer within the enterprise (determined by examining the Internet Address of the requesting process), then the authentication service can be used to load a first service to decrypt the input. However, when the requesting process is executing on a computer outside of the enterprise (determined by examining the Internet Address of the requesting process), then the authentication service can be used to load a second service to decrypt the input.

[0402] The authentication service can access and interact with other services defined in this specification. By way of example, the authentication service can access and interact with the common directory service to query for accessible services, or for entities providing a service. The authentication service can query for general user information.

[0403] The authentication service can access and interact with the services defined in US Patent 5,850,518. By way of example, the authentication service can access and interact with the Thread Directory Service to query for accessible services, or for entities providing a service.

[0404] The authentication service can access and interact with a second authentication service based on the requesting process's Internet Address. By way of example, a first service is configured to access and interact with a first authentication service. When a requesting process accesses and interacts with the first service, then the first service accesses and interacts with the first authentication service. The first authentication service determines the requesting process is executing on a computer within the enterprise (i.e., within a given internet address range), and the first authentication service permits full access to the services provided by the first service. When the first authentication service determines the requesting process is executing on a computer outside of the enterprise, then the first authentication service accesses and interacts with a second authentication service to determine if the requesting process is authenticated. By way of example, the second authentication service may use a challenge response method, which is well known in the state of the industry, to verify that the requesting process has appropriate credentials. Alternatively, the second authentication service may configure the first service to use one or more different components such as a different decryption service. Alternatively, the first second authentication service may restrict access to one or more minor services provided by the first service. Alternatively, the second authentication service may cause the first service to access and interact with a second directory service having a different set of registered services. In this manner, when the requesting process is executing within the enterprise, it can access and interact with a first common directory service, but, when the requesting process is executing on a computer outside of the enterprise, it can access and interact with a second common directory service distinct from the first common directory service. Similarly, if the Internet Address of the requesting process cannot be determined, then a third common directory service distinct from the first and second, can be used. In this manner, we can control access to common directory services based on where the requesting

process executes, how the requesting process communicates with the first service, or based on the information the requesting process provides to the first service.

[0405] As part of the authentication service, the authentication service can access and interact with the common directory service to query for information components. By way of example, if the requesting process provides the authentication service with a unique identifier, the authentication service can access and interact with the common directory service to obtain the registration entry corresponding to the unique identifier. In this manner, the authentication service can configure a service based on the known registration information related to the requesting process.

[0406] In this regard, the authentication service provides more than just examining the client Internet Address to determine if the client is allowed access to the primary service. In our invention, the authentication service provides the capability to:

- authenticate access to the primary service based on the requesting process's Internet Address
- authenticate access to a minor service based on the requesting process's Internet Address
- dynamically configure the components of a service, based on the requesting process's Internet Address
- dynamically configure the components of a service, based on information provided by a requesting process
- access and interact with a common directory service to determine authentication service
- access and interact with a common directory service to determine authentication service to use based on requesting process's network access point

- select the common directory service accessible to the requesting process based on the Internet Address of the computer the requesting process is executing on.

[0407] The authentication service can be implemented to determine the credentials of the requesting process, and determine what service directories should be used to configure the authentication service. By way of example, but not limitation, the authentication service can use reverse domain name lookup to determine the domain name of the requesting process. With that information, the authentication service can then set environment variables, perform initializations, load services, or perform other actions so as to influence the behavior in satisfying the request. In one embodiment, using the Daytona data management system, the environment variable DS_APPS is set to the applications that are permitted (i.e., the associated tables and record class descriptions which collectively define the data being managed). Similarly, the environment variable DS_PATH defines one or more directories to search when looking for the associated service directories (ie., the data being managed).

[0408] When a request is made to connect to a service, the request can be sent as components of information (possibly formatted similar to ksh environment variable rules), and using the requesting processes credentials (ie., the Internet domain name associated with the requesting process connection on the client side), we can query the service directory for environment variables and perform the appropriate initialization. A request, such as a command=query description="report sales for last month" would be queried against a first service directory when coming from a gtlinc domain, whereas the same request sent from a second company with a separate internet domain, would be queried against a second service directory. This method can also be used for registration such that when the request includes:

1. command=register description="payment information" name=payservice

[0409] Then the request will be executed against a service directory identified by the requesting process (client) credentials.

[0410] Generic Front End Loading Service

[0411] A generic front end loader (gfel) is used to initialize an address space for a service, and access and interact with the service. An example generic front end loader is provided in Program Listings 9.1 through 9.4. Parameters are provided to gfel indicating name / value pairs. When a parameter name is given using the keyword primitive then gfel will register the indicated service with the directory service. As an example, using the parameters:

[0412] name=daytime_service

[0413] location=libservices.so.1.0

[0414] primitive=INET

[0415] physical=/local:/tmp/ds_comprim

[0416] will cause gfel to dynamically load the libservices.so.1.0 library, locate the daytime_service module within the library, and start the service listening on a unix domain socket given by the path name /tmp/ds_comprim.

[0417] Alternatively, the parameters

[0418] name=daytime_service

[0419] location=libservices.so.1.0

[0420] primitive=INET

[0421] physical=192.168.20.15:9996

[0422] will cause gfel to dynamically load the libservices.so.1.0 library, locate the daytime_service module within the library, and start the service listening on a inet socket given by internet address 192.168.20.15 port 9996.

[0423] In either case, the service is registered with the directory service.

[0424] When gfel is used without the physical name / value pair, then gfel will establish access and interact with the directory service to determine how to access and interact with the service given by the name= name/value pair. As an example, the specification:

[0425] name=daytime_service

[0426] will cause gfel to access and interact with the common directory service to locate, and to access and interact with the daytime_service.

[0427] Using the location and name parameters together, without the primitive or physical parameters, will cause gfel to dynamically load the service into the current gfel process.

[0428] An implementation can use the common directory service to determine the appropriate actions for each of the name/value pairs provided to gfel. For example, a specification of:

[0429] nvpairs=tds name=route

[0430] will cause the gfel to access and interact with the common directory service to determine a name service that gfel can access and interact with, to determine the appropriate actions for using the specification. In this context, the

name / value pairs appearing in the specification to gfel, other than nvpairs=tds, are not processed by the gfel process itself, but rather, by a service that gfel will access and interact with. Thus, the remainder of the specification to the gfel process represents arbitrary named representations and gfel has no preconceived notion of what the arbitrary named representations represent. When combining this with the binding service of US patent 5,850,518, then gfel can use the binding service to determine what the arbitrary named representations represent. In one implementation, gfel may cause binding methods to be registered with the binding service, and then access and interact with the binding service to determine what the name/value pair represents, and how to process it.

[0431] When gfel is to execute a service, then gfel will examine the service to determine if the service includes an administrative minor service. If so, then gfel will also accept requests from a requesting process to perform administrative capabilities. Note that gfel will typically use two distinct mechanisms for accepting requests in this regard. By way of example, gfel can accept requests from an administrative communication link such as a Unix domain socket accessible only on the computer that gfel is executing on, and accept general requests from a request communication link such as an Internet socket. By way of example, gfel will open a pathname to a unix domain socket such as /usr/share/gfel/engine/admin and accept administrative requests. Similarly, gfel will open a socket using the Internet Address and specified port to accept general requests for the service. In this manner, even while the primary service offered by gfel is executing, we can connect on the administrative link to access and interact with gfel to perform administrative functions, such as examining the state of gfel, examining the historic use data, reconfiguring the service offered, change logging information, redirecting requests, or otherwise alter the basic behavior of the service without having to terminate and restart the service. This could include, for example, changing the Internet Address and/or port that gfel is using for general access and interaction.

[0432] Payment Connection Service

[0433] A consumer registers payment service (CPS) which is executing on consumer computer (CC). The registration is with common directory service (CDS). The registration information includes connectivity requirements and consumer the unique identifier (CID). Connectivity requirements can include one or more of: an Internet Address, Port, protocol, access method, communication mechanism, or other information required for CDS to be able to communicate with CPS. Such communication can be communications communicated via computer mail.

[0434] A service provider registers requesting service (SPRS) which is executing on service provider computer (SPC). The registration is with common directory service (CDS). The registration information includes connectivity requirements and service provider the unique identifier (SPID). Connectivity requirements can include one or more of: an Internet Address, Port, protocol, access method, communication mechanism, or other information required for CDS to be able to communicate with CPS. Such communication can be communications communicated via computer mail.

[0435] SPRS communicates request to CDS. The request is to access and interact with CPS. SPRS provides CDS with SPID and CID.

[0436] CDS registers a transaction in progress and assigns the unique identifier (TID). Registration includes TID, SPID, and CID.

[0437] CDS locates CPS registration, and communicates the unique identifier (TID) to CPS. CPS receives the unique identifier (TID).

[0438] CPS connects to CDS. CPS communicates TID and CID to CDS. CDS locates registration entry for the unique identifier (TID), and CDS facilitates communication from CPS to SPRS. CPS communicates payment information to SPRS.

[0439] A first embodiment, Program Listing 14.0, provides a process service which can be included in an engine configuration specification. In this embodiment, the process service receives the tid from the CDS. It then closes the connection from CDS. It then opens a payment_info file, duplicates the file descriptor as file descriptor 0 which is standard input, and calls gfel to connect to the common directory service having the specified tid. The gfel service will invoke the talk2 service which reads from standard input and sends to the connected service.

[0440] A second embodiment, Program Listing 14.1, provides a process service which can be included in an engine configuration specification. In this embodiment, the process service receives the tid from the CDS, and also the SPID. It then accesses and interacts with the CDS to query for the registration information related to SPID. In then checks for an information component called Service Provider. If the information component is present, it prompts the user to determine if the user wants to accept the communication request from the specified service provider. If the user does not enter yes, then the connection is declined. Otherwise, the request is accepted and CPS calls gfel to proceed as in the first embodiment. In this embodiment, the name of the service provider requesting payment information would be provided to the consumer. The consumer has the choice to accept or decline. Variations of the embodiment could include the use of a graphic display, or a graphic representation being displayed to the user. By way of example, the user could be presented with a graphic representation of ACCEPT and a DECLINE, and then using a pointing device such as a computer mouse, the user could select the desired option. The software component responsive to the mouse click, would then accept or decline

the request for payment information. This could also include displaying the name of the service provider and possibly other registered information related to the service provider. In yet another embodiment, the service provider could communicate the amount due and that information could also be presented to the consumer. In this manner, it would give the consumer a second chance to ensure they agree to the transaction.

[0441] When the consumer is using a computer with a monitor, keyboard, mouse, and means of graphical display, that when the CPS is started, it would display a graphical representation indicating that the CPS is running. In a first implementation, this may include a graphical representation such as a wallet being open. When the CPS terminates, the graphical representation would depict a wallet being closed. Customization could include a graphical icon of a purse being open when CPS is running, and a graphical icon of a purse being closed when CPS is not running. In other implementations, when CPS registers with CDS, it can receive a communication representative of a first graphical representation to display when CPS is running. Similarly, it can receive a second graphical representation to display when CPS is no longer running. Note that if a graphical representation is displayed indicating CPS is no longer running, then a component of software can be responsive to the consumer using a pointing device such as a mouse "click", to cause CPS to start running. In such cases, the graphical representation would then be changed to indicate that CPS is running. In this context, CPS would start executing and would register with CDS. When CPS registers with CDS, it can indicate to CDS that CPS already has graphical representation information and such information would then not need to be provided by CDS.

[0442] A third embodiment, Program Listing 14.2, provides a process service which can be included in an engine configuration specification. In this embodiment, the process service receives the tid from the CDS, as well as various acceptable

payment types to the service provider. In this embodiment, CPS matches the payment types accepted by the service provider to those recorded in the payment_info file accessed by CPS to match up the information requested with the payment information to be provided by CPS. Multiple variations to the embodiment are possible including implementing a preferred payment type by the consumer in which case CPS would determine if the preferred payment type is accepted by the service provider before choosing other payment types. In another variation to the embodiment, a graphical display may appear on the consumer computer monitor (display) indicating one or more matching payment types, and permitting the consumer to select the preferred payment type for that transaction. In yet another variation, the graphical representation of the various payment types available by the consumer could be displayed, and, when matched against those payment types supported by the service provider, the graphical representation could be changed to a second graphical representation, such as highlighting, to indicate that the payment type is acceptable. The consumer could then depress the mouse button to "click" on one of the highlighted graphical representations to indicate which of the payment types the consumer wishes to use.

[0443] In another variation, the CPS could be designed to monitor for communication communicated via computer mail protocol. In doing so, the CPS would register with the CDS that the CDS should communicate pending connections (transactions) to CPS via computer mail. When a computer mail message is received on the CC, the CPS would examine the mail message to determine if it is an appropriate pending transaction communicated from CDS. If so, then CPS would read the unique identifier (TID) and connect to CDS. CPS communicates the unique identifier (TID) and CID to CDS. CDS locates registration entry for the unique identifier (TID), and CDS facilitates communication from CPS to SPRS. CPS communicates payment information to SPRS. Note that once CPS accesses and

interacts with CDS, then SPRS could send to CDS other information components that are required.

[0444] In another embodiment, CPS could be registered with a common directory service wherein CPS acts as a conduit to a second component of software. In this embodiment, the second component of software could access and interact with a database system to query for payment information and provide same to CPS instead of having CPS open and read an accessible file.

[0445] In another embodiment a dual callback system can be used. In this embodiment, SPRS accesses and interacts with CDS to request payment information service for consumer with CID. CDS receives CID and SPID from SPRS. CDS creates a transaction in progress registration and assign a unique identifier (TID). The registration including SPID, CID, and TID. CDS then disconnects from SPRS. CDS uses CID to locate CPS registration, and connects to CPS, and communicates the unique identifier (TID) to CPS. CPS receives the unique identifier (TID). CPS disconnects from CDS and CDS disconnects from CPS. CPS connects to CDS. CPS communicates TID and CID to CDS. CDS locates registration entry for the unique identifier (TID). CDS updates the unique identifier (TID) entry with pending transaction information recording CDS process having CDS connection open. CDS uses SPID of registration entry corresponding to the unique identifier (TID), to locate SPRS entry. CDS connects to SPRS and sends the unique identifier (TID). SPRS receives the unique identifier (TID). SPRS disconnects from CDS and CDS disconnects from SPRS. SPRS calls CDS and sends SPID and TID. CDS, responsive to receiving SPID and TID, locates TID entry. CDS accesses and interacts with pending transaction information of the recorded CDS process having CDS connection open to pass file descriptors to said CDS process. CDS then notifies recorded CDS process to facilitate communications. CPS then communicates payment information to SPRS.

[0446] Note that the method of the payment service can be used to facilitate other such services. By way of example, CPS could be a contact service providing consumer contact information. In such cases, the SPRS would be requesting access to the contact service instead of the payment service for the specified consumer. SPRS could provide to CPS the information component name or names that it is looking for. CPS could then fill in the response. The CDS would facilitate communication just as it does for the CPS providing payment information.

[0447] Alternatively, CPS could be corporate information such as that which would normally appear in a Dunn & Bradstreet (D&B) report. In such cases, the SPRS would be requesting access to the corporate information associated with a particular the unique identifier. Thus, the SPRS could send the desired service type for a particular the unique identifier to CDS, and CDS could locate the service and facilitate the connectivity as described in this specification.

[0448] A Data Sharing Service

[0449] A first process of a first computer of the network accesses and interacts with a directory service to register the first process as providing a particular type of data, such as an Excel spreadsheet template, an Excel spreadsheet formula, an encoded voice stream, a video stream, voice and video stream, genealogy information, medical records information, financial data, or the like. The registration information includes the connectivity required to reach the service. The registration information could also include one or more of the registration information components described in US Patent 5,850,518, such as the input types understood by the service, the output types, or the data representation used in communicating with the service. The first process listens for a request. By way of example, the first process could register a description of "northrup genealogy" and connectivity information of

"elmer.gtlinc.com:9999" where elmer.gtlinc.com is the name of a computer within the gtlinc.com domain, and 9999 represents the port that the first process is listening on. Using standard name services, the registration process can convert the name elmer.gtlinc.com to an Internet Address, or the, the directory service can use the domain name service to determine the Internet Address when needed.

[0450] A second process of a second computer of the network accesses and interacts with the directory service to request access to the first service. By way of example, the second process could provide criteria description="northrup genealogy". The directory service, responsive to receiving the request, locates the first service registration entry and accesses the registration entry. The directory service then facilitates the connectivity to the first service.

[0451] The invention is not limited to data stream processing. The underlying communication could be implemented through various protocols and various communication methods such as through sockets.

[0452] Medical Test Results Reporting Service

[0453] HIPAA (Health Insurance Portability and Accountability Act of 1996) regulations have been put into law which clearly define the treatment of patient information by health care providers. These regulations cover both patient privacy standards as well as security standards that the health care provider must adhere to with respect to digital patient data.

[0454] Medical test results reporting can be automated within the HIPAA regulations via a service. The service can be provided by the health care provider, or by a third party service provider.

[0455] The health care provider summarizes the results of medical tests in a format to be made available to the patient. This may include an image scan of a printed lab report, physician notes, or other means of documentation. In a preferred embodiment, the scan images would be saved in an industry standard file format such that a viewer can be used to view the images (hardware to provide same and software is provided by Hewlett Packard's ScanJet Scanner). This information is then recorded in a data store.

[0456] The medical test results in the data store are encrypted with a digital key that is stored and will be made available only to the patient. Alternatively, just prior to providing the results, the software service will encrypt the data from the data store according to the patient digital key.

[0457] The health care provider notifies the patient that the medical test results are available. Such notification can be via telephone, email, or other means such as software notification.

[0458] The health care provider communicates the unique id of the results to the patient.

[0459] The patient registers with the health care provider service and receives a unique id (PID). This must be completed before the patient can retrieve the test results.

[0460] The patient becomes aware of the availability of the test results. Using the PID and the unique id of the test results, the patient connects to the service and retrieves the medical test results.

[0461] In a first embodiment, the Health Care Provider (HCP) maintains computer (HCC) with communication device. The HCP provides a directory service

(HDS) executing on HCC. HCP registers patient with directory service and patient is assigned a unique identifier (PID). The HCP registers a service to provide lab test results (HCLRS) to patient (PID). The registration is assigned a unique identifier (TID), and the registration records PID. HCP communicates the unique identifier (TID) to patient with PID. Patient with PID uses computer (PCC) with a communication device, to start a first process on PCC. The first process accesses and interacts with HDS. The first process provides PID and TID to HDS. HDS locates the TID entry, and facilitates connectivity to HCLRS. HCLRS, responsive to the connectivity, provides first process with medical test results. The first process uses the digital key known to patient with PID to decrypt the results, and display the results to the patient.

[0462] In a second embodiment, the Health Care Provider (HCP) maintains computer (HCC) with communication device to permit communication with the network. The HCP provides a directory service (HDS) executing on HCC. HCP registers patient with directory service and patient is assigned a unique identifier (PID). Patient with PID maintains and uses computer (PCC) with communication device to permit communication with the network. Patient causes software service PSS to begin executing on PCC. PSS accesses and interacts with HDS to register PSS and connectivity required to reach PSS.

[0463] The HCP registers a service to provide lab test results (HCLRS) to patient (PID). The registration is assigned a unique identifier (TID), and the registration records PID. HCLRS accesses and interacts with HDS, causing HDS to locate PSS entry, access PSS entry, and to access and interact with PSS. HDS provides PSS with the unique identifier (TID). PSS accesses and interacts with HDS, providing HDS with PID and TID. HDS locates the registration entry with PID and TID, and facilitates connectivity to HCLRS. HCLRS, responsive to the connectivity, provides first process with medical test results. The first process uses the digital key

known to patient with PID to decrypt the results, and display the results to the patient.

[0464] In a third embodiment, the Health Care Provider (HCP) maintains computer (HCC) with communication device to permit communication with a network. The HCP uses a component of software to register with a common directory service executing on a second computer of the network. HCP is assigned a unique identifier (HCID).

[0465] Patient uses a computer PCC with communication device to permit access to network. Patient causes a component of software to be executed and patient registers with common directory service. The registration including a unique identifier (PID) uniquely qualifying the patient from other registered patients.

[0466] Patient causes software service PSS to begin executing on PCC. PSS accesses and interacts with common directory service to register PSS and connectivity required to reach PSS.

[0467] The HCP registers a service to provide lab test results (HCLRS) to patient (PID). The registration is assigned a unique identifier (TID), and the registration records HCID and PID. HCLRS accesses and interacts with common directory service, causing common directory service to locate PSS entry, access PSS entry, and to access and interact with PSS. PSS is provided the unique identifier (TID). The access and interaction now complete, and the common directory service disconnects from the communication with PSS.

[0468] PSS accesses and interacts with common directory service, providing common directory service with PID and TID. The common directory service locates the registration entry with PID and TID, and facilitates connectivity to HCLRS.

HCLRS, responsive to the connectivity, provides medical test results. The PSS uses the digital key known to patient with PID to decrypt the results, and display the results to the patient.

[0469] In a fourth embodiment, the Health Care Provider (HCP) maintains computer (HCC) with communication device to permit communication with a network. The HCP uses a component of software to register with a common directory service (CDS) executing on a second computer of the network. HCP is assigned a unique identifier (HCID).

[0470] Patient uses a computer PCC with communication device permitting access to network. Patient causes a component of software to be executed and patient registers with CDS. The registration including a unique identifier (PID) uniquely qualifying the patient from other registered patients.

[0471] Patient causes software service PSS to begin executing on PCC. PSS accesses and interacts with CDS to register PSS, the registration including PID and connectivity required to reach PSS.

[0472] The HCP uses a component of software to register with CDS, a service to provide lab test results (HCLRS) to patient (PID). The registration is assigned a unique identifier (TID), and the registration records HCID and PID.

[0473] CDS locates PSS registration entry having PID and PSS, access the entry, and connects to PSS. CDS communicates the unique identifier (TID) to PSS. CDS disconnects from PSS communication link.

[0474] PSS connects to CDS, and sends PID and the unique identifier (TID).

[0475] CDS, responsive to receiving PID and TID, locates the registration entry with PID and TID, and connects to HCLRS. CDS uses file descriptor passing techniques to pass the file descriptor of HCLRS to PSS.

[0476] HCLRS encrypts medical test results and sends the results to PSS. PSS receives the results, and uses the digital key known to patient with PID to decrypt the results, and displays the results to the patient.

[0477] Alternatively, the health care provider may choose to use a third party to host the reporting service. Using this method, the provider posts the availability notice to the third party provider, who in turn notifies the patient of the availability. When ready to retrieve the results, the patient service connects to the third party service, which in turn then connects to the health care provider. During the ensuing transaction, the patient service is delivered the results of the lab tests.

[0478] Physician Pharmaceutical Service

[0479] A pharmacist uses a computer (PCC) with operating system with interfaces for communication connectivity and synchronization, and a communication device, to execute a component of software which registers pharmacists with common directory service (CDS) running on a second computer of the network. The pharmacist is assigned a unique identifier (PHARMD).

[0480] A doctor uses a computer (DCC) with operating system with interfaces for communication connectivity and synchronization, and a communication device, to execute a component of software which registers doctor with CDS and is assigned a unique identifier DID.

[0481] A patient is registered with CDS and is assigned a unique identifier PID. The patient could uses a computer (HCC) with operating system with interfaces for communication connectivity and synchronization, and a communication device, to execute a component of software which registers patient with CDS and is assigned a unique identifier PID. Alternatively, the doctor or an assistant thereof can register patient with CDS.

[0482] The doctor prescribes a prescription for patient and records the prescription in a data store.

[0483] The doctor uses computer to execute a component of software (MDS) to provide PID prescription information. MDS connects to CDS and registers as a service, the registration including the connectivity required to reach the service, and the DID.

[0484] The patient visits PHARMD office and provides PHARMD with their PID, and their doctor's name (or DID). The pharmacist uses a component of software (COS) on PCC to connect to CDS and request prescription information for patient PID, the request including the DID (or doctor's name).

[0485] CDS registers the request as a pending transaction and assigns the unique identifier (TID), the registration including DID and PID.

[0486] CDS uses DID as criteria to locate MDS registration and connects to MDS. CDS sends TID to MDS. MDS receives TID. CDS and MDS disconnect. MDS connects to CDS and provides TID and DID. CDS locates the unique identifier (TID) entry and facilitates communication to COS. MDS then provides COS with prescription information.

[0487] In a preferred embodiment, the prescription information would be encrypted according to a digital certificate. In this manner, when MDS provides the prescription information, the information would be encrypted. It is noted that COS would need to decrypt the information. In one embodiment, the digital certificate would be that of the pharmacist. In a second embodiment, the digital certificate would be assigned and known to the patient. In a third embodiment, the digital certificate would be known to the doctor. In any case, the doctor software MDS would need to have access to the digital certificate, as would the COS.

[0488] Data Store Forwarding Service

[0489] A challenge with software services is that the corresponding process must also be accessible to the network. There are times, however, when due to power failures, network interruptions, scheduled down time, and the other situations, where the computer or the corresponding process may not always be accessible via the network.

[0490] When the service is to provide a stream of data, it is desirable to offer that data even if the host computer is not accessible. To resolve this limitation, a recording service is provided, along with a playback service.

[0491] This permits a first process of a first computer of the network, to connect to a recording service to record data provided by the first process. The recording service will record the data to a data store, and assign a unique name to the data. By way of example, a unique file name can be used when the data store is a standard file. A playback service, given the unique name to the data, can access and playback the data to a requesting process.

[0492] The recording service can be a first process of a first computer of the network, listening for requests on a network endpoint, such as an Internet Address

and port. The recording service accepts a connection from a requesting process, and records whatever the requesting process sends, to a data store, such as a file. The file is uniquely named. The recording service can be registered with a common directory service running on a second computer of the network. Program Listing 14.3 provides an embodiment of a recording service process for a software engine, or for use with gfel.

[0493] The playback service can be a third process of the first computer of the network, listening for requests on a given network endpoint, such as an Internet Address and port. The playback service accepts a unique name, accesses and interacts with a data store defined by the unique name, and communicates the contents thereof. The playback service can be registered with the common directory service. Program Listing 14.4 provides an embodiment of a recording service process for a software engine, or for use with gfel.

[0494] By connecting to the recording service, a requesting process can retrieve a unique file name, and can send data to be recorded by the recording service. The playback service can be registered with the common directory service. A second requesting process can then connect to the common directory service to locate the playback service, and can provide the playback service with the specified unique file name. The second requesting process would then receive the contents of the data previously recorded by the recording service.

[0495] In an alternative embodiment, the playback service could erase the contents of the data store given by the unique identifier after the playback has occurred. Similarly, the playback service could connect to the common directory service and cause the registration entry for the playback service to be deleted.

[0496] In an alternative embodiment, the playback service can determine the data type by examining the content of the data, in order to determine playback modes. By way of example, this would be comparable to using a mime type to determine the playback software that is to be used.

[0497] Academic Transcript Service

[0498] School grades are considered private information, and cannot be disclosed to third parties. Providing current grades and academic transcripts via the world wide web is less then secure in the current state of the art. To address this concern, an Academic Transcript Service is provided.

[0499] An educational institution uses a computer with communication device and an operating system with interfaces for communication connectivity and synchronization (ACC) to access network.

[0500] A student is registered with common directory service and assigned a unique identifier (SID).

[0501] The academic institution is registered with the common directory service and assigned a unique identifier (AID).

[0502] A student uses a computer with communication device and an operating system with interfaces for communication connectivity and synchronization (SCC) to access network.

[0503] The Academic Institution runs an academic reporting service (ARS) on ACC. ARS registers with common directory service, the registration including connectivity requirements to reach ARS.

[0504] The student executes a component of software (RADAR) on SCC, the component of software designed to request and display academic records. The student provides RADAR with SID. RADAR connects to the common directory service and request academic records for SID. The common directory service receives the request and records SID and AID into a transaction registration entry, the transaction being assigned a unique identifier (TID). CDS connects to ARS and sends the unique identifier (TID). ARS receives TID and both ARS and CDS disconnect from the communication. ARS then connects to CDS and provides AID and TID. CDS, responsive to receiving AID and TID, locates the corresponding transaction entry and facilitates connection to RADAR. ARS provides RADAR with academic transcripts, and RADAR receives and processes the academic transcripts.

[0505] In a second embodiment, student is registered with CDS and assigned SID. The academic institution is registered with CDS and assigned AID. The Academic Institution runs an academic reporting service (ARS) on ACC. ARS registers with common directory service, the registration including connectivity requirements to reach ARS.

[0506] RADAR begins executing on SCC. Student provides RADAR with SID. RADAR registers with CDS, the registration including SID and connectivity required to reach RADAR.

[0507] RADAR connects to CDS and request academic records for SID. The request can include AID or academic institution name which can be used to locate AID. CDS registers the request as a transaction in progress and assigns a unique identifier (TID). The registration entry can include AID. CDS and RADAR then disconnect.

[0508] CDS locates ARS entry, connects to ARS, and sends TID. ARS receives TID. Both CDS and ARS disconnect.

[0509] ARS connects to CDS. ARS sends AID and TID to CDS. ARS receives from CDS, the SID. ARS uses SID to access and interact with datastore having academic transcripts. ARS accesses the transcripts.

[0510] CDS, responsive to receiving AID and TID, locates RADAR registration entry using SID as the lookup value. CDS creates registration entry for active ARS session, and assigns a unique identifier ATID. CDS connects to RADAR and sends RADAR the ATID. RADAR receives ATID. RADAR and CDS then disconnect from the communication link.

[0511] RADAR connects to CDS and sends ATID and SID. CDS, responsive to receiving ATID and SID, locates registration entry and facilitates communication connectivity between RADAR and ARS. ARS then communicates academic transcripts. When complete, RADAR, ARS, and CDS all disconnect from the communications.

[0512] Public Office Election Service

[0513] Many have considered using the Internet for general elections. The belief is that more registered people would participate in the voting if permitted to vote over the Internet, instead of driving to drive to a local school. The challenge, of course, is the lack of security and the mechanisms to institute elections over the Internet. To address this concern, an election service is provided.

[0514] An election office, or appropriate authority, uses a computer with communication device and an operating system with interfaces for communication connectivity and synchronization (ECC) to access network.

[0515] A registered voter is registered with common directory service and assigned a unique identifier (VID).

[0516] The authorizing agency is registered with the common directory service and assigned a unique identifier (EID).

[0517] A voter uses a computer with communication device and an operating system with interfaces for communication connectivity and synchronization (VCC) to access network.

[0518] The authorizing agency runs an election service (ES) on ECC. ES registers with common directory service (CDS), the registration including connectivity requirements to reach ES.

[0519] The voter causes software (VCS) to execute on VCC. VCS connects to CDS and request access to voting information. CDS locates ES registration entry, and facilitates communication connectivity on behalf of VCS to ES.

[0520] ES provides VCS with voting information. The information containing candidate information. The information could contain instructions. The information could contain additional information such as political party, desired office, the term of office, or other such information as would be useful to the voter. Once complete, ES, VCS, and CDS all disconnect from the various communication links.

[0521] VCS requests VID from voter. The voter provides VID to VCS. The voter also selects the desired candidate (either through mouse click, pointing device, touch screen, voice, keyboard, keypad, or other mechanism as one skilled in the state of the art would understand, or via an industry standard method for providing input to a software service).

[0522] VCS connects to CDS and request access to ES. VCS provides CDS with VID. CDS creates a transaction in progress registration entry and assigns a unique identifier (TID). The registration entry including connectivity information required to reach VCS. VCS then disconnects from CDS. CDS connects to ES and provides ES with the unique identifier (TID). ES receives the unique identifier (TID). ES and CDS disconnect.

[0523] ES connects to CDS and provides EID and TID. CDS locates TID entry. CDS uses connectivity information to connect to VCS and provides VCS with TID. VCS receives TID. CDS and VCS disconnect. VCS connects to CDS and provides VID and TID. VCS locates TID entry and facilitates communication connectivity on behalf of VCS to ES. VCS then provides ES with voter supplied information.

[0524] Medical Records Service

[0525] Extensive, accurate and up-to-date medical records may not always be available in times of urgent need. A Medical Records Service provides a means to make an individual's complete medical record available to a health care provider while controlling access and ensuring privacy.

[0526] To use the service, the patient registers with the third party Medical Records Service, creating a common directory service (CDS) entry for the patient and

obtaining a unique identifier (PID). The entry also includes a limited-use personal identifier (LUPID).

[0527] Health care providers interested in using the service also register with CDS, creating a CDS entry and obtaining a unique identifier (HCPID).

[0528] The health care provider registers with CDS, a Health Care Reporting Service (HCRS) executing on health care provider's computer having a communication device and an operating system with interfaces for communication connectivity and synchronization. CDS creates a registration entry and assigns the unique identifier HCRSID.

[0529] When a patient visits a health care provider, the health care provider creates a record in CDS indicating that care has been provided for that patient. Medical records are not stored in CDS, it contains only a record of the relationship between the patient with PID and the provider with HCPID.

[0530] When the medical records for a patient need to be referenced (by an emergency room staff, for example), the patient consents by providing the inquiring party with PID and LUPID. It is noted that the inquiring party must also be registered with CDS and have a unique identifier (IPID).

[0531] The inquiring party uses a component of software (COS) on a computer having a communication device and an operating system with interfaces for communication connectivity and synchronization to request medical records for patient with PID and personal identifier LUPID. CDS receives the request and creates a transaction in progress registration entry, assigning a unique identifier (TID). CDS accesses the registered entries for PID to determine HCPID.

[0532] CDS uses HCPID to lookup the health care provider HCRS service. Once located, CDS connects to HCRS and sends the unique identifier (TID). HCRS receives the unique identifier (TID) and disconnects, as does CDS. HCRS then connects to CDS and provides HCPID and TID. CDS receives HCPID and TID, and locates corresponding registry entry for TID. COS then facilitates communication connectivity with COS. HCRS then sends to COS the records for patient PID.

[0533] In a preferred embodiment, the communicated patient medical records would be encrypted according to a certificate. The certificate would have to be known by either the Health Care Provider and the inquiring party, in order to decrypt the data. In one embodiment, the certificate could be the LUPID, as it is available to all parties. In a second embodiment, the certificate could be the PID, or the HCPID, or the IPID. In any case the certificate for public key encryption or the equivalent thereof, must be known by the corresponding parties.

[0534] Resume Matching Service

[0535] Due to privacy concerns, it is not always desirable to post one's resume on public bulletin boards or job posting sites. Likewise, it is expensive for employers to use employment agencies, classified advertisements and job websites to post job openings. A resume matching service provides a private, secure method of matching job applicants to companies with job openings.

[0536] Individuals register with a third party that provides the service. The registration is anonymous. Registration includes job history, education and other typical data included on a resume.

[0537] Companies register with a third party that provides the service. The registration is not anonymous. Companies provide such information as company background, location, benefits, etc. that are of interest to job seekers.

[0538] When a company has a job opening, the description of the job is posted to the directory service. Details include job title, salary, education requirements, location, start date, etc.

[0539] When individuals wish to search for job openings, they connect to the directory service and indicate availability, along with salary requirements. The resume matching service scans available job postings by companies and matches the job seeker's data to the job opening. Matches are retrieved and sent to the individual for review. The individual scans the job openings, along with the company information posted in the directory service. Each job opening is either rejected or accepted. When a job opening is accepted, the service is contacted, and the individual's resume is sent to the company, along with personal contact information for the individual. When there is a mutual interest, a job interview is scheduled.

[0540] Company Credit Reporting Service

[0541] Obtaining credit information on potential customers is useful prior to establishing credit terms. Although commercial services are available to obtain such information, the cost may be prohibitive for many businesses. An alternative Credit Reporting Service makes this possible.

[0542] Companies register with the third party Credit Reporting Service, creating a directory service entry and obtaining a unique identifier. Registration indicates the company's participation and willingness to share data on their credit history with other companies.

[0543] Companies also register entries in the central directory service indicating those other companies with which they have done business. Companies

contribute their own credit experience with other companies to their own Credit History Service, which can be accessed via the central directory service.

[0544] Third party services provided value-added services such as public records reporting, credit scoring, etc., for a fee for specific queries against the central directory service.

[0545] A Prepay Service

[0546] Various payment methods have been used for electronic commerce. The prepay service is a method for maintaining secure payment information.

[0547] A consumer uses a computer with communication device and an operating system with interfaces for communication connectivity and synchronization to execute an APS component of software. The consumer interacts with APS to provide registration information. APS registers consumer with common directory service (CDS), and consumer is assigned a unique identifier (CID).

[0548] A service provider uses a computer with communication device and an operating system with interfaces for communication connectivity and synchronization to execute a PS component of software. The service provider interacts with PS to provide registration information. PS registers service provider with common directory service (CDS), and service provider is assigned a unique identifier (SPID).

[0549] Service provider causes PS to execute and PS accesses and interacts with CDS to register as a prepay service, the registration including connectivity requirements to reach PS.

[0550] Consumer uses APS to prepay services. APS accesses and interacts with CDS to locate prepay service PS. Consumer specifies the amount of prepaid service desired. Consumer uses payment service described elsewhere in this specification to pay for the prepaid service. By way of example, consumer authorizes $50 prepaid service to be billed to consumer's American Express credit card. The prepay service (PS) receives payment information and causes the consumer's American Express account to be billed $50. The prepay service (PS) registers the credit with a directory service, the registration including the CID, the outstanding credit amount, and a unique identifier (ANID). The prepay service sends the ANID to APS. APS receives the ANID and records in the payment information file a prepaid payment type and account ANID.

[0551] In subsequent uses of the payment service, the service provider receiving the payment information would access and interact with CDS to locate the prepay service. Once located the service provider software would then request a debit to the ANID account for CID. The prepay service would provide service provider with a separate authorizing payment information to bill against. In a preferred embodiment, this would include a mastercard account, expiration date, and cardholder information.

[0552] In an alternative embodiment, the consumer payment service (CPS) would receive the bill amount from SPRS. CPS can access and interact with CDS to locate prepay service and send ANID, CID, and bill amount to prepay service. The prepay service, responsive to receiving ANID, CID, and bill amount, would locate registration entry for ANID and would authorize payment of bill amount to credit card held by service provider. In doing to, the prepay service would communicate the payment information (i.e., card holder, credit card type, credit card number, credit card expiration) to CPS which would then communicate that information to SPRS.

[0553] In an alternative embodiment, the prepay service would be used in place of the CPS. This, however, requires registration with CDS to indicate that prepay service should be used for providing payment service for CID. In such cases, it is preferable for the prepay service to make such registration information available to CDS. Thus, when SPRS request payment information service for CID to CDS, then CDS would record the unique identifier (TID) and communicate the CID and TID to prepay service, and prepay service would validate the CID and provide payment information to SPRS. This would permit the prepay service to provide SPRS with a temporary credit card with a preset limit not to exceed the balance due to the service provider SPID.

[0554] Translation Service

[0555] Language translations such as Japanese to English or vice-a-versa, are often desirable. The google search engine offered at http://www.google.com provides a translation service for cached HTML documents. When a user of the network receives email in a foreign language, there are no translation services via the Internet to provide translation from a first language to a second language. Similarly, there are no services to translates from a first language to a second language when sending email. Yet electronic mail is one of the most widely used services of the Internet.

[0556] A service provider can register with common directory service (CDS) and is assigned a unique identifier (SPID). The service provider provides a language translation service (LTS) component of software on service provider computer (SPCC). The service provider causes LTS to execute on SPCC. LTS registers with CDS. The registration including the connectivity required to reach LTS.

[0557] A consumer can use a component of software (COS) on consumer computer (CC) to register with CDS. The consumer is provided a unique identifier (CID).

[0558] The consumer can use a component of software (SCOS) on consumer computer (CC) to request CDS to connect with a language translation service providing translation from English to Chinese. CDS locates LTS registration entry, and creates a transaction in progress registration entry, assigning a unique identifier (TID). CDS connects to LTS and sends TID to LTS. LTS receives TID and disconnects from CDS, as well as CDS disconnecting from LTS. LTS connects to CDS and provides SPID and TID. CDS locates TID entry and connects LTS to SCOS.

[0559] In this manner, SCOS can communicate information to LTS which is to be translated from English to Chinese. When complete, LTS, SCOS, and CDS all disconnect from the communication.

[0560] Note that in a first embodiment, CDS could provide SCOS with the connectivity required to reach LTS independent of CDS. In a second embodiment, CDS can disconnect after the connection has been made between LTS and SCOS. In a third embodiment the data representation to be communicated to LTS may require translation from a first format to a second format. In this manner, various brokers can be dynamically loaded to provide such translation. By way of example, if SCOS is communicating an unformatted component of an electronic mail message to be translated, and LTS requires the format to be HTML, then a broker service can be used to provide translation for the unformatted text to be formatted according to HTML rules. Similarly, the results of LTS may be communicated in HTML format. Thus, a broker service can be used to provide translation from HTML format to unformatted content.

[0561] An Environment Service

[0562] An environment service starts out as a process essentially representing a vacuum, such as empty space. There are no objects, no services, nor anything of interest in the environment.

[0563] A requesting process having appropriate authorization can connect to the environment service and specify that a service is to be executed within the environment, the service being a controlling service, in which case, the controlling service acts as the administrator of the environment.

[0564] A requesting process having appropriate authorization, can connect to the environment and induce a behavior by requesting a first service to be executed within the environment. The controlling service accesses and interacts with the directory service to locate the desired first service and causes the service to effect the environment. By way of example, this can include loading the service and executing the service as a thread. Alternatively, the controlling service could connect to the first service and communicate with the service. The controlling service registers the first service in an environment directory service (registry).

[0565] A requesting process having appropriate authorization, can connect to the environment and induce a behavior by requesting a second service to be executed within the environment. The controlling service accesses and interacts with the directory service to locate the desired second service and causes the second service to effect the environment. By way of example, this can include loading the service and executing the second service as a second thread. Alternatively, the controlling service could connect to the second service and communicate with the second service. The controlling service registers the second service in an environment directory service (registry).

[0566] The first service and the second service can compete for computing resources, discover each other through querying the environment directory service, and otherwise interact with each other as deemed appropriate. Alternatively, the controlling service can determine the interactions between the first service and the second service, or otherwise assist in their influencing their behavior.

[0567] By way of example, a first service can represent an atom, such as a hydrogen atom. A second service can represent an atom such as an oxygen atom. A third service can represent a second oxygen atom. When the controlling service recognizes the atoms and has means to bind the atoms, then the controlling service can induce a fourth service representative of a water molecule, and cause the first, second, and third service to be suspended, as they are now part of the fourth service. Alternatively, the first, second, and third service may be able to execute, but only within the environment of the fourth service. In such cases, the controlling service would create a new environment and register the first, second, and third service within that environment. By way of example, the controlling service creates a new directory service registry and moves the first, second, and third service registration from the current environment registry to the new directory service registry. The controlling service may also suspend, or otherwise lower the priority values of the services are deemed appropriate. When the embodiment includes multithreading, then the priority value of the thread may be set. When the embodiment includes single threading, then the priority value of the process may be set.

[0568] The controlling service can use Virtual Reality Modeling Language (VRML), which uses the right-handed Cartesian Coordinate System. Accordingly, a first service can have a current location within the environment. Note that VRML is well understood in the state of art. VRML was recognized as an international standard (ISO/IEC-14772-1:1997) by the International Organization for

Standardization (ISO) and the International Electrotechnical Commission (IEC) in December, 1997. Alternatively, as new industry standards for virtual modeling emerge, such standards could be used.

[0569] A service can induce the effect of wind or air movement to change the coordinate of one or more services within the environment. The coordinate of a service within the environment can be maintained with the environment directory service.

[0570] A service can induce the effect of heat or cold. By inducing the effect of heat within a given coordinate range, the service can register the current heat value with the controlling service, which could query the environment registry to determine which services would be effected by the heat, and notify the services accordingly. The controlling service can use multiple services to assist in controlling the environment. By way of example, a temperature service can be a service of the controlling service. When the controlling service receives notification of heat within a given coordinate range, the controlling service can communicate that information to the temperature service which then access the environment registry to determine the effected services.

[0571] A service within the environment can simulate motion. In doing so, the service would have a velocity and a path. The service could update the current coordinates with the environment registry as appropriate. In an alternative embodiment, the service can maintain the current coordinates, and the controlling service could query the service to determine the current coordinates.

[0572] Although alternative embodiments could languages other than VRML, having the standard VRML permits third parties to create services and register the services with the environment service.

[0573] A consumer of the environment service can use a component of software on the consumer computer to connect to the environment and receive the current state of the environment. In such cases, the component of software may need to render graphic images or otherwise understand what the state of the environment, as communicated by the environment service, represents. In an alternative implementation, the component of software could access and interact with a broker service which understands how to interpret the state of the environment, and which can communicate the information to the consumer component of software in a manner understood by the component of software. By way of example, the broker service could convert the output of the environment service to a multimedia presentation and communicate the multimedia presentation to the consumer component of software.

[0574] The implementation does not need to used the atomic level of modeling. By way of example, a virtual landscape such as a virtual mall, a tour, or other landscape could be used as well.

[0575]. The implementation could also be used for genetic sequencing, medical discoveries such as drug interactions, or other types of services in which one needs to understand the interactions between two or more entities within an environment.

[0576] Note that the environment could use ADAM, A Dynamic Attribute Manager, as described in Programming With UNIX Threads, C. Northrup, John Wiley and Sons, ISBN 0-471-13751-0, to implement multithreading of services within the environment. A modification of ADAM as a service is defined elsewhere in this specification.

[0577] Typical Embodiment

[0578] A typical embodiment includes consumer computer, which can be a HP Pavilion running Windows 98, with Internet access via an Internet Service Provider. Internet access is typically via an analog modem for dial-up access, or via high-speed broadband DSL, cable or fixed wireless service. The service provider computer(s), which can be a workgroup class server such as a Sun Enterprise 450 Server running the Solaris operating system, with dedicated access to the Internet via an Internet Service Provider. This access is typically a high-speed service such as Frame Relay, DS-1 or DS-3 service. The service provider computer(s) typically have large amounts of disk storage either internal or in external disk arrays. The directory service computer(s) is typically a midrange system such as a Sun Enterprise 3500 multiprocessor server running the Solaris operating system, configured with dedicated access to the Internet via an Internet Service Provider. This access is typically a high-speed service such as Frame Relay, DS-1 or DS-3 service. The directory service computer(s) typically have large amounts of disk storage either internal or in external disk arrays. The actual computers in use will be determined by processing requirements. In extremely high-volume processing environments clusters of server computers may be used by the service provider or the directory service.

[0579] Figure 1 is a diagram of a computer network communicating according to the present invention. A directory service computer 31 is connected to a service provider computer 23 and a customer's computer 32 via the internet, represented at 37. Figure 1 provides an illustration of such an embodiment. Note that each computer has at least one communication device, such as a modem or an Ethernet card; a monitor display such as a Philips Magnavox; an input device such as a keyboard; a pointing device such as a Microsoft Mouse, or other appropriate mouse for the configuration; an operating system, such as Linux, AIX, HP-UX, Microsoft Windows 98, NT, 2000, XP, or other Microsoft Windows operating system, Solaris,

Irix, Linux, Unix, BSD, Free-BSD, OS/390 or other commercially available operating system for the architecture. Alternatively, the operating system could be one provided by academia, open source, or other such operating system.

[0580] Processing flow embodiments are provided in Figures 2-7, showing the order of the processing to use the invention.

[0581] Figure 2 is a flowchart of a directory service connection service. In step 51, a common directory service (CDS) executes on a directory service computer (31, Fig. 1). The common directory service maintains 52 registry SP, and listens 53 for communication on network endpoint. A service process executes 54 on a service provider computer (32, Fig. 1), and then connects 55 to the common directory service, and sends 56 registration information to the common directory service. CDS creates 58 a registry entry SP-1 in registry SP and assigns a unique identifier SPID. The common directory service sends 62 the SPID to the service process, and the service process receives 63 the SPID, followed by the service process disconnecting 64 from communication. This results in the common directory service disconnecting 66 from communication.

[0582] After the common directory service disconnects 66 from the communication, the service process connects 71 to the common directory service, and sends 72 service registration information, SPID, IP address, and port (SIP) to the common directory service, and the common directory service receives 73 registration information. At this point, the common directory service creates 74 registry entry SPS-1 in the service process and assigns a unique identifier (SPSID). The common directory service sends 76 the SPSID to service process, and the service process receives 77 the SPSID and disconnects 78 from communication. This is followed by the common directory service disconnecting 79 from communication.

[0583] When the common directory service disconnects 78 from communication, the service process executes 81 on the common directory service and listens for communication on IP address and port. A consumer service executes 83 on consumer computer (33, Fig. 1), and connects 84 to the common directory service.

[0584] The common directory service accepts 91 a connection by a consumer service requesting 92 access and interacting with SPSID, receives 94 a request and locates the SPSID registry entry. The common directory service receives 93 the request, then creates 96 a transaction registration entry and assigns a unique identifier (TID), and records 98 SPID, TID, and active connection information from a consumer service CS in entry TID.

[0585] The common directory service connects 101 to an IP address and port of SPSID, and the service process accepts 102 the connection. The common directory service then sends 103 the unique identifier (TID) to the service process. The service process receives 104 the unique identifier (TID), disconnects 105, and the common directory service disconnects 106. The service process connects 111 to the common directory service, the common directory service accepts 112 connection, and the service process sends 113 the unique identifier (TID) and SPID. The common directory service then receives 114 the unique identifier (TID) and SPID, locates 115 the transaction entry, and connects 116 the common directory service connection from service process to active connection from CS.

[0586] Figure 3 is a flowchart of a directory service use. As can be seen, the common directory service executes 131 on the directory service computer (31, Fig. 1). The common directory service maintains 132 registry service process, and listens 133 for communication on network endpoint.

[0587] Figure 4 is a flowchart of a service provider registration. A service process (SP) executes 151 on the service provider computer (32, Fig. 1), connects 152 to the common directory service, and sends 153 registration information to the common directory service. The common directory service receives 154 registration information, and creates 155 registry entry SP-1 in service process and assigns the unique identifier (SPID). The common directory service then sends 156 SPID to service process. The service process receives 157 SPID, disconnects 158 from communication, and the common directory service disconnects 159 from communication.

[0588] Figure 5 is a flowchart of a service registration. The service process connects 171 to the common directory service, sends 172 service registration information SPID, IP address, and port (SIP) to the common directory service. The common directory service receives 173 registration information, creates 174 registry entry SPS-1 in registry and assigns the unique identifier (SPSID), and sends 175 SPSID to service process. The service process receives 176 SPSID and disconnects 177 from communication. The common directory service disconnects 178 from communication and the service process executes 179 on the common directory service and listens for communication on IP address and port

[0589] Figure 6 is a flowchart of a consumer registration. A consumer process executes 191 on the consumer computer (33, Fig. 1), connects 192 to the common directory service, and sends 193 registration information to the common directory service. The common directory service then receives 194 registration information, creates 195 registry entry CID-1 in service process, assigns the unique identifier (CID), and sends 196 the CID to consumer process. The consumer process receives 197 the CID and disconnects 198 from communication, and the common directory service disconnects 199 from communication.

[0590] Figure 7 is a flowchart of a consumer request for service. A consumer process executes 221 on consumer computer (33, Fig. 1). A service request process executes 222 on the directory service computer (31, Fig. 1). The consumer process connects 223 to the common directory service, and the common directory service accepts 224 the connection. The consumer process then requests 225 access and interaction with SPSID. The common directory service receives 226 the request and locates SPSID registry entry, registers 227 the transaction registry entry and assigns the unique identifier (TID), and records 229 the SPID and TID in registry entry. The common directory service maintains 230 the connection with consumer process, connects 231 to an IP address and port of the SPSID. The service process accepts 233 the connection and the common directory service sends 234 the unique identifier (TID) to the service process. The service process receives 235 the unique identifier (TID) and disconnects 236. The common directory service disconnects 237 and the service process connects 238 to the service request process. The service request process accepts 241 the connection, and the service process sends 242 the TID and SPID to the service request process. The service request process then receives 243 the TID and SPID, locates 245 a transaction entry, and communicates 247 communication from service request process to the common directory service maintained connection with consumer process.

[0591] In a preferred embodiment, a prototype table is created containing a msg indicator along with a flds indicator and a description of the columns for the table. The prototype table can also include one or more rows. The Daytona DC-rcd command can be used to generate the data dictionary information. For example, using "DC-rcd SERVICES > rcd.SERVICES" will generate the data dictionary information for us, without having to enter that information manually. Three examples of a service registries are given in Program Listings 16.1, 16.2 and 16.3, respectively. The command to generate the data dictionary is shown in Program Listing 16.4. The resulting generated data dictionary is shown in Program Listing

16.5. The Daytona Synop command can be used for data dictionary reporting. Alternatively, the backtalk command shipped with daytona can be used to generate data dictionary information.

[0592] Program Listing 16.6 shows a second embodiment of the service registry prototype table. Using the DC-rcd command, the data dictionary shown in Program Listing 16.7 is then generated. Similarly, the embodiment of a providers registry is shown in Program Listing 16.8, with the generated data dictionary in Program Listing 16.9. An embodiment to register an entry is given in Program Listing 16.10, while Program Listing 16.11 provides an embodiment to report registration entry information. The embodiment could use the Daytona Tracy command to process the Daytona query, which can understand either Cymbal, SQL, or a combination thereof.

[0593] Note that in Program Listing 16.12, the registration request is given as a Daytona task (also called a function/predicate/procedure, or fpp). Semantically, the idea is that there is some goal that a task is intended to achieve, and the code that is has for doing that is free to call its own private helper fpps as well as other tasks. Using Daytona's Tracy command, the fpp is converted to C source code, which can then be compiled into object code. In normal processing, the object code is then linked with the appropriate Daytona runtime objects and libraries to generate an executable program. Alternatively, the object code can be linked with other application object code to provide the fpp directly at the application level. By way of example, an application programmer can write their own source code which can then invoke the desired fpp by linking with the object code, and other Daytona runtime objects and libraries. In an alternative embodiment, an application process can use the invention to call the fpp by dynamically loading the fpp according to the specification of this invention. The application service, however, will need to ensure that the Daytona Sizup command is executed as appropriate to maintain the Daytona data files and indices. The use of the Daytona code synthesis (code generation) permits the administrative capabilities of registration, query, delete, modification,

replication, reporting, and other such functionality as would be required in administering and managing the data, to be instrumented through the methods and systems of this specification.

[0594] In an embodiment shown in Figure 8, the service directory would be horizontally partitioned. A horizontal partition divides the rows of the service directory (registry) horizontally based on criteria and put each group in its own file. The resultant individual files will be easier to manage. Another benefit is that the physical field that would have previously been recorded in the registry can be eliminated, thus saving disk storage. In Figure 8, the horizontal partition is the category of the service. In Figure 9 the horizontal partition is based on the provider. In Figure 10, the horizontal partition is based on the activity. In Figure 11, the horizontal partition is based on the cost. In Figure 12, the horizontal partition is based on the protocol. In Figure 13, the horizontal partition is based on the entity type.

[0595] If the underlying data management system supports horizontal partitioning, then such partitioning techniques could be used as well.

[0596] The Directory Service

[0597] The Directory Service (TDS) can administer one or more Service Directories (SD). In the most primitive form, a Service Directory contains one or more entries representing entities providing a service. Each service directory is uniquely named. A service directory entry is comprised of one or more Information Components (IC) given as name/value pairs, as depicted in Figure 14. The primitive operations for TDS include register, query, and delete. Additional administrative operations are supported, such as index, update, modify, and replicate.

[0598] Figure 14 illustrates a typical TDS instance. In this illustration, there are three service directories being maintained by a single TDS process. Figures 15 and 16 are diagrams illustrating different implementations of TDS instances. Figure 15 is a sample configuration for System sol27 (Solaris 2.7). Figure 16 is a sample configuration using multiple operating systems and different OS implementations. In Figure 16, three implementations of Unix, one implementation of Microsoft Windows and one implementation of Linux each have a TSD instance and are interconnected.

[0599] Different entities provide different types of services, although a single entity can provide a multitude of services. A component of software, for example, can provide some form of a service. The term component of software is deliberately chosen to imply that less then an entire executable program can still provide a service. Examples include objects from shared libraries, a specification for an interpretative language, a device, a process, and even a thread of execution. The operating system itself can be said to provide a service, or a multitude of services.

[0600] A service provided by a component of software can be registered in TDS. When needed, a separate process can cause the service to be started. "The Connection Service", described in US Patent 5,850,518, describes one technique for registering components of a service.

[0601] A user can provide a service. Consider, for example, the Netscape Navigator, or Microsoft IE. Both of these programs require a user to enter a URL in order to determine what to display next. Thus, the user provides input and this is considered a service. Similarly, an email application stores email directed towards a specific user. Retrieving the email is considered a service.

[0602] Service providers provide services, and consumers consume services. A consumer, however, can also provide a service. Similarly, a service can also consume services.

[0603] In generalized terms, a service is facilitated by a process. For example, a spell checker is a process that provides a service. Similarly, a caching process can provide a service. The distinction of when a process is a service, and when it is a consumer, is relative to what the process is doing at a particular point in time.

[0604] In the context of TDS, a process can be heavyweight, medium-weight, or lightweight. A process can consist of multiple threads of execution, including kernel threads.

[0605] Each entity is referred to as a point of communication (compoint). To facilitate the method, each compoint can participate in a communication with another compoint. A compoint can either send a communication, receive a communication, or both send and receive communications. A communication can be sent as messages, data, and streams.

[0606] The generalization of services permits a single TDS to administer multiple service directories. This provides maximum flexibility in organizing service entries. Note, however, that multiple TDS processes can execute on the same system. Furthermore, remote TDS processes can broadcast their availability and this will cause the local TDS to register the remote as an entity providing a service.

[0607] In a typical environment, a system wide TDS is available as a compoint. The system wide TDS provides a default service directory for a specific system. Be careful not to confuse the term system wide with network wide, or corporate wide. The term system wide simply means a TDS that is executing on a single computer

and is available to any compoint executing on that computer. The system wide TDS is also available for remote processes.

[0608] All request received that do not specify a particular service directory, will be executed against the default service directory. The default service directory contains one or more service type entries. Each entry is composed of one or more IC pairs (name/value pairs).

[0609] The system wide TDS can maintain multiple service directories. This permits the grouping of common service entries into a service directory dedicated to the service type. Each service directory has a unique identifier.

[0610] An example TDS is shown in figure TDS-2 for a system called sol27. In this example, TDS maintains a default service directory, an application services service directory, and a process service directory.

[0611] When TDS is started, it will broadcast its availability. This permits a TDS on one system to share information with a TDS on a second system. When a local TDS receives a broadcast from a remote TDS, the local TDS will query the remote TDS to learn its registered characteristics. As long as the characteristics can be determined, the local TDS will register the remote TDS in the local TDS's default service directory, as an entity providing a service.

[0612] An environment with 5 systems, each running their own TDS and sharing information is shown in figure TDS-3. Each of the TDS process's broadcast their availability.

[0613] Each service directory has a record-class-description (rcd) defining the IC pairs for the service type entry. Record class descriptions are described in more detail in section 2.2 and 2.3 of this document.

[0614] A service entry consists of multiple IC pairs. The service entry has an assigned the unique identifier. Each IC pair consists of a name and a value. The grammar is given as:

[0615] service type entry : id name=value [name=value] ... [name=value]

[0616] A value can contain white space provided it is quoted. The following examples show various name / value pairs.

[0617] tds=default

[0618] tds="system wide service directory"

[0619] tds='application specific service directory'

[0620] All entries within a given service directory must be unique. Uniqueness, however, can be a single IC pair. Thus, the following are considered unique entries:

[0621] name=tds physical=/local:/usr/lib/share/TDS/tds_compoint

[0622] name=tds physical=sol28:9998

[0623] name=tds physical=sol28:127.0.0.1:998

[0624] An IC name has attributes describing its use. A private attribute, for example, instructs TDS not to report the IC pair in a query operation.. The default public attribute, however, indicates that the IC pair is to be reported in query operations. Note that a query operation can use the IC name value pair as part of the criteria for selecting the entry, but TDS will not include that IC pair in the query

response. A service directory can also be marked as private, and thus the name of the service directory will not appear in the results of query operations.

[0625] When a first rcd is replaced with a second rcd, the operation can specify a load map to map the existing entries according to the new rcd.

[0626] TDS permits IC pairs to be prefixed with their corresponding service directory identifier. For example, a query command can reference the name IC from the suppliers service directory and the name IC from the products service directory by specifying:

[0627] query supplier.name="GTL.*" product.name=*"

[0628] A record class description (rcd) defines the characteristics of the IC pairs for a given service directory. Each service directory has a rcd.. The rcd defines the IC pairs and their data representation. An example rcd is given as:

```
[0629]  command=rcd                    \
[0630]  sd="applications"          \
[0631]  service_name=str(50)           \
[0632]  registration_date=yymmdd       \
[0633]  value=float                \
[0634]  count=int                  \
[0635]  flag=short                 \
[0636]  provider=str(*)
```

[0637] To impose a rcd, an administrative process must register the rcd with TDS when the service directory is created. Alternatively, a default rcd can be identified through the configuration file. A rcd can be inherited from a parent Service Directory.

[0638] When a process registers a service with TDS, then TDS will search for an applicable rcd and will invoke the corresponding rcd function. Similarly, when the process queries TDS for an accessible service, TDS will search for an applicable rcd and will invoke the corresponding rcd function.

[0639] When a service directory is referenced without an existing rcd, then TDS simply adds the IC pairs are necessary, to the service directory. As an example, the following register command will create the service directory process, and add the pid and uid IC pairs.

[0640] command=register sd=process pid=19452 uid=12345

[0641] This makes TDS lightweight enough for even the simplest of applications. Of course, once a service directory has been created in this fashion, you cannot add a record class description without applying some form of conversion.

[0642] It may be inappropriate to use TDS in this manner for production environments, as there is no provision for validating the registration. Using a record class description, however, will limit registration requests to only those IC pairs defined in the record class description. Additionally, indexing and data management is much more robust when a record class description is defined.

[0643] The primitive operations for TDS include register, query, and delete. Several additional operations are provided for administrative support. Each request to TDS includes a command, and one or more IC pairs, given as parameters. Examples include

[0644] command=register name=tds physical=/local:/usr/TDS/tds_compoint

[0645] command=register name=tds physical=sol28:127.0.0.1:998

[0646] command=query name="*" action=match

[0647] command=query name="this is a string" action=casecmp

[0648] Note that for query command, there is an implied AND operator between the IC pairs. Explicit Boolean operators are also supported. Support for Boolean operators is dependent on the rcd implementation.

[0649] The query operation will report all public IC pairs for the registered service. To limit the scope of the report, a special action IC pair can be used. Assigning a value of match to the action will cause query to report only those IC pairs specified as parameters to the query operation.. The special value of "*" for an IC pair, indicates to match anything.. Thus, the query operation below will report the all entries having a name=Jane and having an email IC component.

[0650] query name="Jane" email="*" action=match

[0651] Multiple action IC pairs can be specified. Valid actions include:

[0652]	strcasecmp	ignore case when comparing
[0653]	numericcmp	use a numeric comparison instead of a ASCII comparison

[0654] The query command supports regular expression pattern matching.. The following query will match on all entries with a name IC pair wherein the value starts with the letter J.

[0655] query name="J*" email="*" action=match

[0656] When using a query command against a single service directory, you can specify the service directory name with a sd parameter given as an IC pair. When using a query command to query multiple service directories, you can prefix the IC pair name with the name of the applicable service directory. Consider for example a

service directory identified as suppliers, and a service directory identified as products.. The following queries are acceptable through TDS.

[0657] command=query sd=suppliers name="Global Tech.*"

[0658] command=query suppliers.name="Global Tech.*" products.name=uwin

[0659] command=query products.name=uwin

[0660] The first registration command, given below, creates a new service entry in the service directory.. The second registration command adds the IC pair primitive=INET to that entry.

[0661] command=register name=tds physical=sol28:9998

[0662] command=register name=tds physical=sol28:9998 primitive=INET

[0663] Using the register command, a process can register a NULL value for an IC pair, thus eliminating it from the service directory.. The service directory does not retain any NULL valued IC pairs. Consider, as an example, the following:

[0664] command=register name=tds physical=sol28:9998 pid=1956

[0665] command=register name=tds physical=sol28:9998 pid=

[0666] In this example, the first operation creates a service directory entry with name=tds physical=sol28:9998, and pid=1956.. The second operation then assigns a NULL value to pid, and thus pid is removed from the entry. (TDS silently discards NULL value IC pairs).

[0667] TDS supports the Cymbal 4th generation language in command statements.. The format is:

[0668] command=DS spec=specification

[0669] TDS provides administrative services for authentication and communication encryption. Administrative services are dynamically re-configurable, and provide sufficient flexibility to meet most needs.

[0670] The authentication service provides for authentication of requesting processes.. The unscramble service provides unscrambling (decryption) of communicated data, and the scramble service offers scrambling (encryption) of communicated responses.

[0671] Administrative services can be registered for a particular service directory, and default to administrative services registered for the system wide service directory. Administrative services can be limited to particular primitives, such as the register primitive, or, registered for all primitives.

[0672] To register an administrative service for a service directory, you must specify the service and the service directory to which it applies. For example:

[0673] command=register service=authentication \

[0674] sd=default name=default_logging location=libservices.so.1.0 physical=-

[0675] To register an administrative service for a particular primitive within a service directory, you would specify the primitive, the service, and the service directory. For example:

[0676] command=register primitive=register service=authentication \

[0677] sd=default name=default_logging location=libservices.so.1.0 physical=-

[0678] Registered administrative services are retained by TDS through the backed data management system.

[0679] Note that the user id of the process that started TDS can replace or otherwise alter the registered administrative services.. Thus, the user id becomes the administrator of TDS.

[0680] The authentication service, if registered, is provided with connection information indicating the system from which the requesting process originates, and possibly the process identifier.. The authentication service provides a status result, which if zero, indicates that authentication is successful. Otherwise, authentication fails and the connection is closed.

[0681] The unscramble service, if registered, is provided with the content.. The unscramble service will unscramble the content and provide a response which is then used for subsequent operations. As implied, the entire message received by TDS cannot be scrambled.. The reason is that TDS must be able to ascertain the service directory component, and possibly the command component IC pairs in order to determine the appropriate unscramble service.

[0682] The scramble service, if registered, is provided with the response communication.. The service will scramble the content, and provide the response to TDS, which then makes it available to the requesting process.

[0683] TDS can be started from /etc/rc services, or, by any application having appropriate privilege.. The first call to TDS will create a default system wide service directory for general registration.. The system wide SD can be disabled by changing the systemSD=default to systemSD=none, in the TDS configuration file. See the section Configuring TDS for more details.

[0684] TDS is configured to recognize and process a set of commands. Nonetheless, a process can register new commands, alter existing commands, and change the behavior of commands. A command is sent to TDS as a name / value pair, with one or more parameters given as IC pairs. Note that IC pairs command=value and tds=value are non-alterable and processed by TDS.. The remainder of the IC pairs are given as parameters to the service corresponding to the command. TDS uses the tds=value IC component to select the appropriate registry service directory. Once located, the register service is called with a reference to the service directory, and the remainder of the IC pairs given as parameters.

[0685] command=register

- [tds=service directory]
- [name=value]

[0686] command=query

- [tds=service directory]
- [name=value]

[0687] command=delete

- [tds=service directory]
- [name=value]

[0688] command=register

- rcd=rcd
- [tds=service directory]
- [location=rcd service location]
- [physical=physical connectivity]
- [inheritence=on|off]

[0689] command=delete

- rcd=rcd_name
- tds=service directory

[0690] TDS provides a registration feature for service such that the administrator of the service directory can register alternative primitive commands.. This includes the register, query, and delete primitives. In registering an alternative command, TDS will change to the owner user identifier during the request.. Thus, if TDS is started by a first user id, and an authorized process registers an alternative query command, then TDS will set the effective user id to the authorized process user id prior to executing the command.. This option can be disabled through the TDS configuration file.

[0691] TDS also permits registration of additional primitives beyond the standard TDS primitives. When TDS receives a command, it will look-up the command name, and execute the specified registered command. To ensure security, however, TDS will temporarily switch to the specified user id when executing the specified command.. This option can be disabled through the TDS configuration file.

[0692] In our network, we provide a supplier service directory and an applications service directory as the default directory services offered through TDS.. The supplier service directory records all suppliers of services while the applications service directory records available application services.. The record class descriptions are given as:

[0693] rcd=Supplier sd=Suppliers

[0694] Name=string(50) Address=string(50) City=string(20)

[0695] State=string(3) Zip=string(10) Contact=string(20)

[0696] Phone=string(10) Id=string(15)

[0697] rcd=Applications sd=Applications

[0698] Name=string(20) Location=string(256) Physical=string(25)

[0699] Primitive=string(10) System=string(15) Release=string(5)

[0700] Os=string(15) Description=string(250) Id=string(15)

[0701] The following services are then registered on the sol27 system.

[0702] command=register sd=Suppliers

[0703] Name="GTL Inc" Address="15 Spring St" City=Princeton State=NJ

[0704] Zip=08542 Contact=sales Phone=(609)924-7305

[0705] Id=123456789

[0706] command=register sd=Applications

[0707] Id=123456789 Name=queued Location=services

[0708] Physical=sol27:9990 Primitive=inet System=sol27

[0709] Os=solaris Description="queue service"

[0710] For our winntsp6 system, we register:

[0711] command=register sd=Suppliers

[0712] Name="GTL Inc" Address="15 Spring St" City=Princeton State=NJ

[0713] Zip=08542 Contact=sales

[0714] Phone=(609)924-7305 Id=123456789

[0715] command=register sd=Applications

[0716] Id=123456789 Name=url_pe Location=services

[0717] Physical=*:* Primitive=inet System=winntsp6

[0718] Os="Windows NT" Description="URL Processing Element"

[0719] Similarly, for the redhat6.1 system we register:

[0720] command=register sd=Suppliers

[0721] Name="GTL Inc" Address="15 Spring St" City=Princeton State=NJ

[0722] Zip=08542 Contact=sales Phone=(609)924-7305

[0723] Id=123456789

[0724] command=register sd=Applications

[0725] Supplierid=123456789 Name=url_pe Location=services

[0726] Physical=*:* Primitive=inet System=winntsp6

[0727] Os="Red Hat Linux" Description="URL Processing Element"

[0728] Once the service entries have been registered, our rcd functions record the entries into indexed files for subsequent retrieval.

[0729] On the sol27 system, we execute a urld process.. This process fetches an HTML page from the Internet, and stores that page on the local system.

[0730] The urld process will query TDS to locate an available url_pe service to process the fetched page.

[0731] Program Listing 1.1 Source Code Listing of One Implementation for the Replacement recv Function

```
1.  #include <netdb.h>
2.  #include <sys/types.h>
3.  #include <sys/socket.h>
4.  #include "services.h"
5.  #include "3d.h"
6.  struct hostent * gethostbyname(const char *name)
7.  {
8.  struct hostent *h = (struct hostent *) _gethostbyname(name);
9.          if( ! h )    {
char *physical=malloc(4096);
char *cp = malloc(4096);
tds_query_p(cp,name,4096);
tds_get_nlpp(cp,NULL,NULL, &physical,NULL,NULL);
14.                 if(physical)                {
char *service, *computer;
service=computer=physical;
while(*service && *service != ':') ++service;
if( *service ) { *service='\0'; ++service; }
return( (struct hostent *) _gethostbyname(name));
}
}
return(h);
}
```

Program Listing 2.0 Engine Service engine.c

```
#include <ast.h>
#include <sfio.h>
#include <stdio.h>
#include <fcntl.h>
#include <dlfcn.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
static int (*startup)(int, char *, void **);
int (*shutdown)(void);
////////////////////////////////////////////////////////////////////
```

```
[0759]  //    the engine components, in the order called [if defined]
[0760]  //          authorize
[0761]  //          input
[0762]  //          preprocess
[0763]  //          process
[0764]  //          postprocess
[0765]  //          response
[0766]  ////////////////////////////////////////////////////////////////
[0767]  static int (*authorize)(int, char *, void **);
[0768]  static int (*input)(int, char *, void **);
[0769]  static int (*preprocess)(int, char *, void **);
[0770]  static int (*process)(int, char *, void **);
[0771]  static int (*postprocess)(int, char *, void **);
[0772]  static int (*response)(int, char *, void **);
[0773]  extern char *getnvpair(char *, char **, char **);
[0774]  extern int   readline(int,char *,int);
[0775]  static int
[0776]  configure_engine(int fd, char *args, void **handle)
[0777]  {
[0778]  char  *e;
[0779]  char  *f="engine.conf";
[0780]  char  buffer[PIPE_BUF];
[0781]  e=getenv("ENGINE_CONFIG");
[0782]  if( access(e,R_OK) == 0) f=e;
[0783]  fd=open(f,O_RDONLY);
[0784]  if( fd <  0 ) { fprintf(stderr,"failed to open config file [%s]\n", f); return(-1); }
[0785]  while( readline(fd,buffer,PIPE_BUF) ) {
-       char *component=NULL;
-       char *name=NULL;
-       char *location=NULL;
-       char *cp=buffer;
-       while( *cp ) {
-       char *n=NULL, *v=NULL;
-       cp=getnvpair(cp, &n, &v );
-       if( ! n || !v ) break;
-       if(strcmp(n,"component")==0) component=v;
-       else if(strcmp(n,"name")==0) name=v;
-       else if(strcmp(n,"location")==0) location=v;
-       }
```

```
-   if( name && location && component ) {
-   void *h=NULL;
-   int (*f)(int,char *,void **)=NULL;
-   h=dlopen(location, RTLD_LAZY);
-   if(!h) continue;
-   f=dlsym(h,name);
-   if(!f) { dlclose(h); continue; }
-   if( strcmp(component,"authenticate")==0) authorize=f;
-   else if(strcmp(component,"input")==0) input=f;
-   else if(strcmp(component,"preprocess")==0)  preprocess=f;
-   else if(strcmp(component,"process")==0) process=f;
-   else if(strcmp(component,"postprocess")==0) postprocess=f;
[0786]  else if(strcmp(component,"response")==0) response=f;
-   else if(strcmp(component,"startup")==0)     startup=f;
-   else if(strcmp(component,"shutdown")==0) shutdown=f;
-   else dlclose(h);
-   }
[0787]  }
[0788]  close(fd);
[0789]  return(0);
[0790]  }
[0791]  static int initialized;
[0792]  static void *handle;
[0793]  int engine(int fd, void *args)
[0794]  {
[0795]  int rc=-1;
[0796]  if( ! initialized ) {
[0797]  initialized=configure_engine(fd, args, handle);
-   if( startup ) startup(fd,args,&handle);
-   if( shutdown ) atexit(shutdown);
[0798]  }
[0799]  if( authorize && (rc=authorize(fd,args,&handle))) goto response;
[0800]  if( input && (rc=input(fd,args,&handle))) goto response;
[0801]  if( preprocess  && (rc=preprocess(fd, args, &handle)))  goto response;
[0802]  if( process && (rc=process(fd,args,&handle))) goto response;
[0803]  if( postprocess && (rc=postprocess(fd,args,&handle))) goto response;
[0804]  response:
[0805]  if( response ) response(fd,args,(void **)rc);
[0806]  return(0);
[0807]  }
```

[0808] Program Listing 2.1 Engine Service getnvpair.c

```
[0809] ////////////////////////////////////////////////////////////
[0810] //    given an input string, return the next name=value pair
[0811] //    the value of name is returned in the location pointed to by name,
[0812] //    and the value is returned at the location pointed to by value.
[0813] //    RETURN:
[0814] //          name has a pointer to the name component, which is null
[0815] //               terminated.  this is an inline replacement, so
[0816] //               this means that the string pointed to by cp is
[0817] //               is modified.
[0818] //          value has a pointer to the value component, which is
[0819] //               null terminated.  this is an inline replacement.
[0820] ////////////////////////////////////////////////////////////
[0821] #include <ast.h>
[0822] #include <sys/types.h>
[0823] #include <stdio.h>
[0824] char *
[0825] getnvpair( char *cp, char **name, char **value)
[0826] {
[0827] int use_quote=0;
[0828] if( ! cp ) return(NULL);
[0829] if( *cp == '#' ) return(NULL);
[0830] while( isspace(*cp)) ++cp;
[0831] *name=cp;
[0832] while( *cp && *cp != '=') ++cp;
[0833] if( *cp == '=') {
-    *cp='\0';
-    *value = ++cp;
-    if( **value == '"') use_quote='"';
-    if( **value == '\'') use_quote='\'';
-    if( use_quote ) { ++cp; *value=cp; }
-    while( *cp ) {
-    if( use_quote && *cp == use_quote)
[0834] { *cp = '\0'; ++cp; break; }
-    else if(!use_quote && isspace(*cp)) break;
-    ++cp;
-    }
-    if( isspace(*cp)) { *cp='\0'; ++cp; }
```

[0835] }

[0836] return(cp);

[0837] }

[0838] Program Listing 2.2 Engine Service authorize.c

[0839] #include <stdio.h>

[0840] int authorize(int fd, char *args, void **handle)

[0841] { return(0); }

[0842] Program Listing 2.3 Engine Service input.c

[0843] #include <stdio.h>

[0844] int input(int fd, char *args, void **handle)

[0845] { return(0); }

[0846] Program Listing 2.4 Engine Service postprocess.c

[0847] #include <stdio.h>

[0848] int postprocess(int fd, char *args, void **handle)

[0849] { return(0); }

[0850] Program Listing 2.5 Engine Service preprocess.c

[0851] #include <stdio.h>

[0852] int preprocess(int fd, char *args, void **handle)

[0853] { return(0); }

[0854] Program Listing 2.6 Engine Service process.c

[0855] int process(int fd, char *args, void **handle)

[0856] { return(0);}

[0857] Program Listing 2.7 Engine Service response.c

[0858] int response(int fd, char *args, void **handle)

[0859] { return(0); }

[0860] Program Listing 2.8 Engine Service readline.c

```
[0861]  #include <ast.h>
[0862]  #include <stdio.h>
[0863]  #include <sfio.h>
[0864]  #include <hash.h>
[0865]  #include <memory.h>
[0866]  #include <errno.h>
[0867]  int
[0868]  readline(int fd, char *buffer, size_t len)
[0869]  {
[0870]  ssize_t      n, rc;
[0871]  char c=0, oldc, *ptr;
[0872]  char  *in_ptr=buffer;
[0873]  int p;
[0874]  ////////////////////////////////////////////////////
[0875]  //      try the fast way first.  using peek_c, we can see
[0876]  //      if there is a new-line in the input stream, and
[0877]  //      we can read up through the new-line.
[0878]  ////////////////////////////////////////////////////
[0879]  n=0;
[0880]  do {
-       ////////////////////////////////////////////////
-       //      wait for up to 30 seconds for something to
-       //      be readable.
-       ////////////////////////////////////////////////
-       wait_read(fd,30);
-       p=peek_c(fd,'\n');
-       ////////////////////////////////////////////////
-       //      if this is interactive, then peek_c will fail
-       //      with -1 because we cannot peek on a tty.
-       //      Instead, we have to break out of this loop, and
-       //      read one byte at a time.
-       //
-       //      NOTE: we really should be testing to see if this
-       //            is a tty, or not.
-       ////////////////////////////////////////////////
-       if( p < 0 ) break;
-       if(p && p < len) {
-       int r=0;
-       r=read(fd,in_ptr,p+1);
-       if(r>2) {
                1. p-=r;
                2. n+=r;
```

```
                3. if((buffer[r-2]=='\\') && (buffer[r-1]=='\0'))
                4. in_ptr=in_ptr+r;
                5. else break;
        -   }
        -   else if(r>0) {
                1. if( buffer[r-1]=='\n') buffer[r-1]='\0'; n=r-1;  break; }
        -   else if(r==0) return(n);
        -   }
        -   if(p==0) break;
[0881]      } while(1);
[0882]      if(n) return(n);
[0883]      oldc=0;
[0884]      ptr=buffer;
[0885]      for(n=1; n < len; n++ ) {
        -   again:
        -   oldc=c;
        -   if( (rc=read(fd,&c, 1)) == 1 ) {
        -   *ptr = c;
        -   ++ptr;
        -   if( c == '\n' ) {
                1. if(oldc == '\\' ) { --ptr; --ptr; continue; }
                2. --ptr;
                3. break;
        -   }
        -   } else if (rc == 0 ) { if( n == 1 ) return(0); else break; }
[0886]      else { if( errno == EINTR ) goto again; return(-1);}
[0887]      }
[0888]      *ptr = 0;
[0889]      return(n);
[0890]      }
```

[0891] Program Listing 2.9 Engine Service wait_read.c

```
[0892]  #include <stdio.h>
[0893]  #include <stdlib.h>
[0894]  #include <sys/select.h>
[0895]  #include <sys/time.h>
[0896]  #include <dlfcn.h>
[0897]  #include <errno.h>
[0898]  int
[0899]  wait_read(int fd, int timeout)
[0900]  {
```

```
[0901]      fd_set      read_fs, error_fs;
[0902]      int    nfds, r;
[0903]      int    adminfd=-1;
[0904]      struct  timeval t;
[0905]      int    (*function)(void) = NULL;
[0906]      memset( &t, '\0', sizeof(struct timeval));
[0907]      FD_ZERO( &read_fs );
[0908]      FD_ZERO( &error_fs );
[0909]      FD_SET( fd, &read_fs );
[0910]      FD_SET( fd, &error_fs );
[0911]      nfds = fd+1;
[0912]      while( 1 )
[0913]      {
            -  t.tv_sec = timeout;
            -  r=select(nfds,&read_fs,NULL,&error_fs,&t);
            -  if( r < 0 ) return( -1 );
            -  else if( r == 0 ) return( 0 );
            -  else if( FD_ISSET(fd,&read_fs) || FD_ISSET(fd,&error_fs))
            -  return(fd);
            -  else if( adminfd != -1 && FD_ISSET(adminfd, &read_fs))
            -  return(adminfd);
[0914]      }
[0915]      return(fd);
[0916]      }
```

[0917] Program Listing 2.10 Engine Service - peek.c

```
[0918]      #include <stdio.h>
[0919]      #include <sys/ioctl.h>
[0920]      #include <errno.h>
[0921]      int
[0922]      peek(int fd)
[0923]      {
[0924]      int n=0;
[0925]      if( ioctl(fd, FIONREAD, &n) != 0) {
            -  fprintf(stderr,"ioctl fd=[%d] failed errno=%d\n",  fd, errno);
            -  return(-1);
[0926]      }
[0927]      return(n);
```

[0928] }

[0929] Program Listing 2.11 Engine Service peek_c.c

```
[0930] #include <errno.h>
[0931] #include <stdio.h>
[0932] #include <sys/ioctl.h>
[0933] #include <errno.h>
[0934] #include <malloc.h>
[0935] #include <sys/socket.h>
[0936] int
[0937] peek_c(int fd, int c)
[0938] {
[0939] int n=0;
[0940] int o=-1;
[0941] int rc=-1;
[0942] int offset=-1;
[0943] static int bsize;
[0944] static char *buffer;
[0945] if( ioctl(fd, FIONREAD, &n) != 0)      {
-      fprintf(stderr,"ioctl fd=[%d] failed errno=%d\n",  fd, errno);
-      return(-1);
[0946] }
[0947] if( n == 0 ) return(-1);
[0948] if( n > bsize ) {
-      if(buffer) free(buffer);
-      buffer=malloc(n+1);
-      bsize=n+1;
[0949] }
[0950] memset(buffer,'\0',bsize);
[0951] rc=recv(fd,buffer,n,MSG_PEEK|MSG_DONTWAIT);
[0952] if(rc > 0 ) { register char *cp=buffer; while( *cp && *cp != c && --rc )  ++offset; }
[0953] return(offset ? offset  : -1);
[0954] }
```

[0955] Program Listing 2.12 Engine Service main.c

```
[0956] #include <stdio.h>
```

[0957] int

[0958] main(int argc, char *argv[])

[0959] {

[0960] engine(0,NULL);

[0961] return(0);

[0962] }

[0963] Program Listing 2.13 Engine Service - Makefile

[0964] :PACKAGE: ast

[0965] CFLAGS = -D_BLD_ast $(CC.DLL) -Wall

[0966] engine :LIBRARY: engine.c readline.c getnvpair.c wait_read.c peek.c peek_c.c -ldl

[0967] Program Listing 2.14 Engine Service - engine.mk

[0968] :PACKAGE: engine ast dl

[0969] enginetest :: main.c

[0970] Program Listing 2.15 Engine Service - dummy.mk

[0971] :PACKAGE: ast

[0972] CFLAGS = -D_BLD_ast $(CC.DL) -Wall

[0973] dummy :LIBRARY: authorize.c input.c postprocess.c preprocess.c process.c response.c

[0974] Program Listing 2.16 Engine Service - engine.conf

[0975] component=authenticate name=authenticate location=../authenticate/libauthenticate.so.1.0

[0976] component=input name=input location=./libdummy.so.1.0

[0977] component=preprocess name=preprocess location=./libdummy.so.1.0

[0978] component=process name=process location=./libdummy.so.1.0

[0979] component=postprocess name=postprocess location=./libdummy.so.1.0

[0980] component=response name=response location=./libdummy.so.1.0

[0981] Program Listing 2.17 Engine Service - engine2.c

[0982] int engine(int fd, void *args)

[0983] {

[0984] int rc=-1;

```
[0985]  static int initialized;
[0986]  configure_engine(fd, args, handle);
[0987]  if( ! initialized )
[0988]  {
[0989]  if( startup ) startup(fd,args,&handle);
[0990]  if( shutdown ) atexit(shutdown);
[0991]  }
[0992]  if( startup || shutdown )
[0993]  initialized=1;
[0994]  if( authorize && (rc=authorize(fd,args,&handle))) goto response;
[0995]  if( input && (rc=input(fd,args,&handle))) goto response;
[0996]  if( preprocess  && (rc=preprocess(fd, args, &handle)))  goto response;
[0997]  if( process && (rc=process(fd,args,&handle))) goto response;
[0998]  if( postprocess && (rc=postprocess(fd,args,&handle))) goto response;
[0999]  response:
[1000]  if( response ) response(fd,args,(void **)rc);
[1001]  return(0);
[1002]  }
```

[1003] Program Listing 2.18 Engine Service - engine3.c

```
[1004]  #include <ast.h>#include <sfio.h>#include <stdio.h>#include <fcntl.h>#include <dlfcn.h>#include
<unistd.h>#include <stdlib.h>#include <errno.h>struct _components {     int (*function)(int, char *, void **);     struct _components
    *next;
[1005]  struct _components    *last;
[1006]  };
[1007]  struct _engine {
[1008]  struct _components *head;
[1009]  struct _components *tail;
[1010]  };
[1011]  struct _engine engineparts;
[1012]  static int (*startup)(int, char *, void **);
[1013]  int (*shutdown)(void);
[1014]  static int (*authorize)(int, char *, void **);
[1015]  static int (*input)(int, char *, void **);
[1016]  static int (*preprocess)(int, char *, void **);
```

```
[1017]  static int (*process)(int, char *, void **);
[1018]  static int (*postprocess)(int, char *, void **);
[1019]  static int (*response)(int, char *, void **);
[1020]  extern char *getnvpair(char *, char **, char **);
[1021]  extern int   readline(int,char *,int);
[1022]  struct _components *
[1023]  Allocate(void)
[1024]  {
[1025]  struct _components *component;
[1026]  component=malloc(sizeof(struct _components));
[1027]  if( component ) memset(component,'\0',sizeof(struct _components));
[1028]  if( ! engineparts.head ) {
        a. engineparts.head=component;
        b. engineparts.tail=component;
[1029]  }
[1030]  else {
        a. component->last = engineparts.tail;
        b. engineparts.tail->next=component;
        c. engineparts.tail=component;
[1031]  }
[1032]  return(component);
[1033]  }
[1034]  static int
[1035]  configure_engine(int fd, char *args, void **handle)
[1036]  {
[1037]  char  *e;
[1038]  char  *f="engine.conf";
[1039]  char  buffer[PIPE_BUF];
[1040]  e=getenv("ENGINE_CONFIG");
[1041]  if( access(e,R_OK) == 0) f=e;
[1042]  fd=open(f,O_RDONLY);
[1043]  if( fd < 0 ) {
        a. fprintf(stderr,"failed to open config file [%s]\n", f);
        b. return(-1);
[1044]  }
[1045]  while( readline(fd,buffer,PIPE_BUF) ) {
        a. char *component=NULL;
        b. char *name=NULL;
```

```
c.    char *location=NULL;
d.    char *cp=buffer;
e.    while( *cp ) {
f.    char *n=NULL, *v=NULL;
g.    cp=getnvpair(cp, &n, &v );
h.    if( ! n || !v ) break;
i.    if( strcmp(n,"component")==0) component=v;
j.    else if(strcmp(n,"name")==0) name=v;
k.    else if(strcmp(n,"location")==0) location=v;
l.    }
m.    if( name && location && component ) {
n.    void *h=NULL;
o.    struct _components *part;

p.    int (*f)(int,char *,void **)=NULL;

q.    h=dlopen(location, RTLD_LAZY);
r.    if(!h) continue;
s.    f=dlsym(h,name);
t.    if(!f) { dlclose(h); continue; }
u.    if(strcmp(component,"startup")==0) startup=f;
v.    else if(strcmp(component,"shutdown")==0) shutdown=f;
w.    else { part=Allocate(); part->function=f; }
x.    }
[1046]      }
[1047]      close(fd);
[1048]      return(0);
[1049]      }
[1050]      static int initialized;
[1051]      static void *handle;
[1052]      int engine2(int fd, void *args)
[1053]      {
[1054]      struct _components *component;
[1055]      if( ! initialized ) {
a.    initialized=configure_engine(fd, args, handle);
b.    if( startup ) startup(fd,args,&handle);
c.    if( shutdown ) atexit(shutdown);
[1056]      }
[1057]      component=engineparts.head;
[1058]      while( component ) {
a.    if( (*component->function)(fd,args,&handle) ) break;
b.    component=component->next;
[1059]      }
[1060]      return(0);
```

[1061] }

[1062] Program Listing 3.0 authentication service - authenticate.c

```
[1063] #include <ast.h>
[1064] #include <sfio.h>
[1065] #include <stdio.h>
[1066] #include <sys/stat.h>
[1067] #include <unistd.h>
[1068] #include <regex.h>
[1069] #include <sys/socket.h>
[1070] #include <sys/types.h>
[1071] #include <netinet/in.h>
[1072] extern const char *inet_ntop(int, const void *, char *, size_t );
[1073] int
[1074] sock_authenticate(int fd, void *handle)
[1075] {
[1076] char  *f="authenticate.conf";
[1077] char  *e=getenv("AUTH_CONF");
[1078] char   buffer[PIPE_BUF];
[1079] struct sockaddr_in *sa=(struct sockaddr_in *) handle;
[1080] const char  *c;
[1081] int    fdi=-1;
[1082] memset(buffer,'\0',PIPE_BUF);
[1083] inet_ntop(AF_INET,&(sa->sin_addr),buffer,sizeof(buffer));
[1084] c=buffer;
[1085] if(e) f=e;
[1086] fdi=open(f,O_RDONLY);
[1087] if(fdi != -1) {
a.  char buffer[PIPE_BUF];
b.  while(readline(fdi,buffer,PIPE_BUF)>0) {
c.  char  *cp=buffer;
d.  char  *host=NULL;
e.  char  *command=NULL;
f.  regex_t     regex;
g.  while( *cp ) {
        i.  char *n=NULL, *v=NULL;
       ii.  cp=getnvpair(cp,&n,&v);
      iii.  if(!n || !v ) break;
```

```
                iv.   if(strcmp(n,"host")==0)
                 v.   host=v;
                vi.   else if(strcmp(n,"command")==0)
               vii.   command=v;
        h.  }
        i.  if( regcomp(&regex,host,REG_EXTENDED|REG_NOSUB)==0)
        j.  {
                 i.   if(regexec(&regex,c,0,NULL,0)==0) {
                ii.   close(fdi);
               iii.   if(command) {
                iv.   if(strcasecmp(command,"DENY")==0)
                              1.   return(1);
                 v.   if(strcasecmp(command,"ALLOW")==0)
                              1.   return(0);
                vi.   }
               vii.   return(0);
              viii.   }
        k.  }
        l.  }
[1088]      }
[1089]      if(fdi != -1 ) close(fdi);
[1090]      return( -1 );
[1091]      }

[1092]      int
[1093]      authenticate(int fd, void *handle, void *args)
[1094]      {
[1095]      struct      stat  statbuffer;
        a.  int   rc;
[1096]      rc=fstat(fd, &statbuffer);
[1097]      if( rc )      return(-1);
[1098]      if( S_ISSOCK(statbuffer.st_mode)) return(sock_authenticate(fd,handle));
[1099]      return(0);
[1100]      }
```

[1101] Program Listing 3.1 Authentication Service - log.h

```
[1102]      typedef struct _log_f_t {
[1103]      int   (*message)(char *, int );
[1104]      int   (*enter)(char *,int);
[1105]      int   (*exit)(char *,int);
[1106]      } log_f_t;
```

[1107] extern log_f_t log_f;

[1108] Program Listing 3.2 Authentication Service - tds2.h

[1109] #include "log.h"

[1110] #define TDS_PRIVATE 0x00000001

[1111] #define TDS_PUBLIC 0x00000002

[1112] #define TDS_MATCH0x00000004

[1113] typedef struct _command_t

[1114] {

a. char *alias;
b. char *service;
c. char *name;
d. char *location;
e. int (*function)(int, char *);

[1115] }

[1116] command_t;

[1117] typedef struct _schema_t

[1118] {

a. char *name;
b. char *location;
c. char *physical;
d. int (*delete)(char *);
e. int (*query)(char *, char *, int, int *);
f. int (*registration)(char *);

[1119] }

[1120] schema_t;

[1121] typedef struct _sd_id_t

[1122] {

a. char *name;
b. char *location;
c. char *physical;
d. char *id;
e. unsigned long status;
f. schema_t *schema;

[1123] }

[1124] sd_id_t;

[1125] typedef struct _sd_base_t

[1126] {

a. Hash_table_t *sd_table;
b. Hash_table_t *command_table;

```
        c.  long        status;
        d.  long        count;
        e.  int         initialized;
[1127]  }
[1128]  sd_base_t;
[1129]  typedef struct _sd_t
[1130]  {
        a.  Hash_table_t    *ste_table;
        b.  long        status;
        c.  sd_id_t         sd_id;
        d.  Hash_table_t    *command_table;
[1131]  }
[1132]  sd_t;
[1133]  typedef struct _ste_t {
[1134]  Hash_table_t    *ic_table;
[1135]  long        status;
[1136]  sd_t        *sd;
[1137]  sd_id_t         ste_id;
[1138]  } ste_t;
[1139]  typedef struct _ic_t {
[1140]  char  *name;
[1141]  char  *value;
[1142]  long  status;
[1143]  ste_t *ste;
[1144]  sd_id_t ic_id;
[1145]  } ic_t;
[1146]  typedef struct _ic_f_t {
[1147]  ic_t  *(*constructor)(char *, char *);
[1148]  } ic_f_t;
[1149]  typedef struct _sd_f_t {
[1150]  sd_t  *(*constructor)(char *);
[1151]  int    (*destructor)(int, char *);
[1152]  char   *(*query)(int, char *);
[1153]  char  *(*register_ste)(int , char *);
[1154]  } sd_f_t;
[1155]  typedef struct _ste_f_t {
[1156]  ic_t   *(*addic)(ste_t *,ic_t *,long);
[1157]  ste_t *(*constructor)();
```

[1158] int (*destructor)(ste_t *);

[1159] } ste_f_t;

[1160] extern ste_f_t ste_f;

[1161] extern ic_f_t ic_f;

[1162] extern sd_f_t sd_f;

[1163] extern sd_base_tsd_base;

[1164] Program Listing 3.3 Authentication Service - makefile

[1165] :PACKAGE: ast

[1166] CCFLAGS = -D_BLD_ast $(CC.DLL) -Wall

[1167] authenticate :LIBRARY: authenticate.c inet_ntop.c getnvpair.c readline.c wait_read.c peek_c.c -lsocket

[1168] Program Listing 3.4 authentication Service - authenticate.conf

[1169] host=127.0.0.1 command=deny severity=error

[1170] host=192.168.15.* command=deny severity=notice

[1171] host=* command=deny severity=notice

[1172] Program Listing 4.1 - Thread Directory Service - tds3.c

[1173] #include <ast.h>

[1174] #include <sfio.h>

[1175] #include <stdio.h>

[1176] #include <hash.h>

[1177] #include <malloc.h>

[1178] #include <string.h>

[1179] #include <sys/utsname.h>

[1180] #include <sys/socket.h>

[1181] #include <netdb.h>

[1182] #include <errno.h>

[1183] #include <dlfcn.h>

[1184] #include <limits.h>

[1185] #include <unistd.h>

[1186] #include "tds2.h"

[1187] static sd_base_t sd_base;

[1188] #if ! defined(TDS_FIRST)

```
[1189]      #    define TDS_FIRST    0x00000001
[1190]      #    define TDS_DEFAULT      0x00000002
[1191]      #endif
[1192]      extern char *getnvpair(char *, char **, char **);
[1193]      static int preprocess_request(char *, sd_t **, ste_t **, int);
[1194]      static char *make_string(char *m)
[1195]      {
[1196]      char *cp=NULL;
[1197]      int  l=0;
[1198]      if(m) {
        a.  for(cp=m;  *cp; ++cp)        ++l;
        b.  l += 1;
        c.  if((cp=malloc(l)) )
        d.  { memset(cp,'\0',l); memcpy(cp,m,l); }
[1199]      }
[1200]      return(cp);
[1201]      }
[1202]      static sd_t *
[1203]      AllocateSD(void)
[1204]      {
[1205]      sd_t  *sd;
[1206]      if( (sd = malloc(sizeof(sd_t))) != NULL) memset(sd,'\0',sizeof(sd_t));
[1207]      return(sd);
[1208]      }
[1209]      static char *
[1210]      Constructor(char *cp)
[1211]      {
[1212]      sd_t  *sd=NULL;
[1213]      ste_t *ste=NULL;
[1214]      ic_t   *ic;
[1215]      if( ! cp ) return(NULL);
[1216]      if( preprocess_request(cp, &sd, &ste, 1) == -1 )
        a.  fprintf(stderr,"Constructor: preprocessor failed\n");
[1217]      if( (ic=(ic_t *)hashget(ste->ic_table,"sd")) == NULL) {
        a.  fprintf(stderr,"Constructor missing service directory name for new SD\n");
        b.  return(NULL);
[1218]      }
[1219]      sd->sd_id.name  = make_string(ic->value);
```

[1220] if((ic=(ic_t *)hashget(ste->ic_table,"location")) != NULL)

a. sd->sd_id.location = make_string(ic->value);

[1221] if((ic=(ic_t *)hashget(ste->ic_table,"physical")) != NULL)

a. sd->sd_id.physical = make_string(ic->value);

[1222] if(++(sd_base.count) == 1) sd->status = TDS_DEFAULT;

[1223] sd->sd_id.id = make_string(sd->sd_id.name);

[1224] hashput(sd_base.sd_table, sd->sd_id.id,sd);

[1225] return(sd->sd_id.id);

[1226] }

[1227] #define TDS_NAME_PRIVATE 0x00000001

[1228] #define TDS_LOCATION_PRIVATE 0x00000002

[1229] #define TDS_PHYSICAL_PRIVATE 0x00000004

[1230] #define TDS_TDS_PRIVATE 0x00000008

[1231] static int

[1232] preprocess_request(char *cp, sd_t **sd, ste_t **ste, int code)

[1233] {

[1234] char *n=NULL;

[1235] char *v=NULL;

[1236] char *sd_id=NULL;

[1237] char *cp2;

[1238] char *name_id=NULL;

[1239] char *location_id=NULL;

[1240] char *physical_id=NULL;

[1241] char *action=NULL;

[1242] int size=0;

[1243] long isprivate=0;

[1244] ic_t *ic=NULL;

[1245] *ste = ste_f.constructor();

[1246] cp=(char *)getnvpair(cp,&n,&v);

[1247] while(n && v)

[1248] {

a. isprivate=0;

b. for(cp2=n; *cp2; ++cp2)

c. if(*cp2 == '.')

i. if(strcasecmp(cp2,".private")==0)

ii. { isprivate=TDS_NAME_PRIVATE; *cp2='\0'; }

d. if(strcasecmp(n,"name")==0) name_id=v;

e. else if(strcasecmp(n,"location")==0) location_id=v;

```
f.    else if(strcasecmp(n,"physical")==0) physical_id=v;
g.    else if(strcasecmp(n,"sd")==0) sd_id=v;
[1249]  else if(strcasecmp(n,"action")==0)
a.    { action=v; goto branchit; }
b.    else if(strcasecmp(n,"command")==0)  goto branchit;
c.    ic=ic_f.constructor(n,v);
d.    ste_f.addic(*ste,ic,isprivate);
[1250]  branchit:
a.    n=NULL;
b.    v=NULL;
c.    cp=(char *)getnvpair(cp,&n,&v);
[1251]  }
[1252]  if(!sd_id) {
a.    ic=ic_f.constructor("tds","default");
b.    ste_f.addic(*ste,ic,isprivate);
c.    ic=ic_f.constructor("sd","default");
d.    ste_f.addic(*ste,ic,isprivate);
e.    *sd=(sd_t *)hashget(sd_base.sd_table,"default");
f.    if(*sd) fprintf(stderr,"using default sd=%x %s\n",*sd, (*sd)->sd_id.id);
g.    else *sd=AllocateSD();
[1253]  }
[1254]  else *sd=(sd_t *)hashget(sd_base.sd_table,sd_id);
[1255]  if(! *sd) {
a.    char *private=NULL;
b.    for(private=sd_id; *private; ++private) if(strcmp(private,".private")==0) break;
c.    *sd=AllocateSD();
d.    if( private ) { *private='\0'; (*sd)->status |= TDS_NAME_PRIVATE; }
e.    (*sd)->sd_id.id=make_string(sd_id);
f.    fprintf(stderr,"preprocess: failed to locate sd\n");
[1256]  }
[1257]  if( code )
[1258]  { if(! (*sd)->ste_table) (*sd)->ste_table=hashalloc(NULL,HASH_name,"STETABLE",0); }
[1259]  else if( ! (*sd)->ste_table)
[1260]  { fprintf(stderr,"the sd->ste_tables is NULL. return -1\n");     return(-1); }
[1261]  if(name_id)       (*ste)->ste_id.name     = make_string(name_id);
[1262]  if(location_id)    (*ste)->ste_id.location = make_string(location_id);
[1263]  if(physical_id)    (*ste)->ste_id.physical = make_string(physical_id);
[1264]  if(action && strcasecmp(action,"match")==0) (*ste)->status = TDS_MATCH;
[1265]  size=100;
[1266]  if(name_id && location_id && physical_id)
a.    size=strlen(name_id)+strlen(location_id)+strlen(physical_id)+4;
```

```
[1267]      (*ste)->ste_id.id=malloc( size );
[1268]      memset((*ste)->ste_id.id,'\0',size);
[1269]      if(name_id && location_id && physical_id)
        a.  sfsprintf((*ste)->ste_id.id,size,"%s|%s|%s",name_id,location_id,physical_id);
[1270]      else if(name_id)
        a.  sfsprintf((*ste)->ste_id.id,size,"%s",name_id);
[1271]      return(0);
[1272]      }
[1273]      static void *
[1274]      LoadCommand(char *name, char *location)
[1275]      {
[1276]      void *handle=NULL;
[1277]      void *f=NULL;
[1278]      if( location ) handle=dlopen(location,RTLD_LAZY);
[1279]      f=dlsym(handle,name);
[1280]      return(f);
[1281]      }
[1282]      static int
[1283]      RegisterCommand(Hash_table_t *command_table,char *alias, char *name,char *location, void *function)
[1284]      {
[1285]      command_t *command;
[1286]      if( ! function ) if(!name) return(-1);
[1287]      command=(command_t *)malloc(sizeof(command_t));
[1288]      memset(command,'\0',sizeof(command_t));
[1289]      command->name=make_string(name);
[1290]      if( alias   ) command->alias=make_string(alias);
[1291]      if( location ) command->location=make_string(location);
[1292]      command->function=function;
[1293]      if( ! function && location ) command->function=LoadCommand(name,location);
[1294]      hashput(command_table, command->name, command);
[1295]      return(0);
[1296]      }
[1297]      static int Service(int fd, char *cp)
[1298]      {
[1299]      ste_t       *ste=NULL;
[1300]      sd_t        *sd=NULL;
```

```
[1301]      Hash_table_t    *command_table=NULL;
[1302]      void        *f=NULL;
[1303]      preprocess_request(cp, &sd, &ste, 1);
[1304]      if(ste ) {
        a.  char *service=NULL, *name=NULL, *location=NULL, *sd_id=NULL;
        b.  ic_t  *ic;
        c.  void  *f;
        d.  ic=(ic_t *)hashget(ste->ic_table,"service");
        e.  if( ic ) service=ic->value;
        f.  ic=(ic_t *)hashget(ste->ic_table,"name");
        g.  if( ic ) name=ic->value;
        h.  ic=(ic_t *)hashget(ste->ic_table,"location");
        i.  if( ic ) location=ic->value;
        j.  ic=(ic_t *)hashget(ste->ic_table,"sd");
        k.  if( ic ) sd_id=ic->value;
        l.  if(sd_id && strcmp(sd_id,"sd_base")==0)
        m.  command_table=sd_base.command_table;
        n.  else
        o.  command_table=sd->command_table;
        p.  if( command_table ) {
        q.  f=LoadCommand(name,location);
        r.  if(f) RegisterCommand(command_table,service,name,location,f);
        s.  }
[1305]      }
[1306]      return(f ? 0 : -1);
[1307]      }
[1308]      static char *
[1309]      Register(int fd, char *cp)
[1310]      {
[1311]      ste_t *ste=NULL;
[1312]      sd_t  *sd=NULL;
[1313]      char  *rc=NULL;
[1314]      preprocess_request(cp, &sd, &ste, 1);
[1315]      if(sd && ste ) {
        a.  if( hashget(sd_base.sd_table, sd->sd_id.id) == NULL)
        b.  hashput(sd_base.sd_table,sd->sd_id.id,sd);
        c.  ste->sd=sd;
        d.  hashput(sd->ste_table,ste->ste_id.id,ste);
        e.  rc=ste->ste_id.id;
[1316]      }
[1317]      return(rc);
[1318]      }
```

```
[1319]      static int
[1320]      compare_ste(int fd, ste_t *ste, ste_t *ste_params)
[1321]      {
[1322]      ic_t *ic_param, *ic;
[1323]      Hash_position_t *hp_params=NULL;
[1324]      Hash_bucket_t  *hb_params=NULL;
[1325]      int    match=1;
[1326]      if( ste_params->ste_id.id && strlen(ste_params->ste_id.id) > 0)
        a.  if(strcmp(ste_params->ste_id.id, ste->ste_id.id)==0) return(1);
[1327]      hp_params=hashscan(ste_params->ic_table,0);
[1328]      while( (hb_params=hashnext(hp_params)) != NULL) {
        a.  char *ic_param_name;
        b.  ic_param_name=hashname(hb_params);
        c.  ic_param=(ic_t *)hashget(ste_params->ic_table,ic_param_name);
        d.  ic=(ic_t *)hashget(ste->ic_table,ic_param_name);
        e.  if(!ic) match=0; break;
        f.  if(strcmp(ic_param->value,"*") == 0 )
        g.  { match=1; continue; }
        h.  if(strcmp(ic_param->value,ic->value)) { match=0; break; }
[1329]      }
[1330]      hashdone(hp_params);
[1331]      return(match);
[1332]      }
[1333]      static int
[1334]      display_ste(int fd, ste_t *ste, ste_t *ste_params)
[1335]      {
[1336]      Sfio_t      *t;
[1337]      char  buffer[2048];
[1338]      int    b;
[1339]      ic_t  *ic=NULL;
[1340]      Hash_position_t *hp_params=NULL;
[1341]      Hash_bucket_t  *hb_params=NULL;
[1342]      t=sftmp(SF_UNBOUND);
[1343]      hp_params=hashscan(ste->ic_table,0);
[1344]      while( (hb_params=hashnext(hp_params)) != NULL) {
        a.  char *ic_param_name;
        b.  ic_t *ic_param=NULL;
        c.  ic_param_name=hashname(hb_params);
        d.  ic_param=(ic_t *)hashget(ste->ic_table,ic_param_name);
        e.  ic=(ic_t *)hashget(ste->ic_table,ic_param_name);
```

```
f.    if( ste_params->status & TDS_MATCH ) {
g.    ic_param=(ic_t *)hashget(ste_params->ic_table,ic_param_name);
h.    if(! ic_param)     ic=NULL;
i.    }
j.    if( ic && !(ic->status & TDS_NAME_PRIVATE) )
k.    sfprintf(t,"%s=%s ", ic->name, ic->value);
[1345]      }
[1346]      hashdone(hp_params);
[1347]      sfseek(t,0L,0);
[1348]      while( (b=sfread(t,buffer,2048)) > 0) write(fd,buffer,b);
[1349]      write(fd,"\n",1);
[1350]      sfclose(t);
[1351]      return(0);
[1352]      }
[1353]      static char * QuerySTE( int fd, Hash_table_t *ste_table, ste_t *ste_params)
[1354]      {
[1355]      Hash_position_t *hp=NULL;
[1356]      Hash_bucket_t  *hb=NULL;
[1357]      char        *rc=NULL;
[1358]      hp=hashscan(ste_table,0);
[1359]      if(hp) {
a.    while( (hb=hashnext(hp)) != NULL) {
b.    char *ste_name=hashname(hb);
c.    ste_t *ste;
d.    ste = (ste_t *) hashget (ste_table, ste_name);
e.    if(fd == 0 ) fd=2;
f.    if( compare_ste( fd, ste, ste_params ) == 1 )
g.    { display_ste(fd, ste, ste_params); rc=ste->ste_id.id; }
h.    }
[1360]      }
[1361]      return(rc);
[1362]      }
[1363]      static char * Query(int fd, char *cp)
[1364]      {
[1365]      ste_t *ste_params=NULL;
[1366]      sd_t  *sd=NULL;
[1367]      char  *rc=NULL;
[1368]      if( preprocess_request(cp, &sd, &ste_params, 1) == -1 ) return(NULL);
[1369]      if(!sd || ! ste_params || ! sd->ste_table) return(NULL);
[1370]      if( strcmp(sd->sd_id.id,"*") ==0) {
```

a. Hash_position_t *hp=NULL;
b. Hash_bucket_t *hb=NULL;
c. hp=hashscan(sd_base.sd_table,0);
d. if(!hp) return(NULL);
e. while((hb=hashnext(hp)) != NULL) {
f. char *sd_name=hashname(hb);
g. sd = (sd_t *) hashget (sd_base.sd_table, sd_name);
h. if(!sd) continue;
i. if(! sd->ste_table) sd->ste_table=hashalloc(NULL,HASH_name,"STE",0);
j. if(sd->status & TDS_NAME_PRIVATE) continue;
k. rc=QuerySTE(fd, sd->ste_table, ste_params);
l. }
m. hashdone(hp);

[1371] }

[1372] else { if(! sd->ste_table) sd->ste_table=hashalloc(NULL,HASH_name,"STE",0);

[1373] rc=QuerySTE(fd, sd->ste_table, ste_params);

[1374] }

[1375] ste_f.destructor(ste_params);

[1376] return(rc);

[1377] }

[1378] static int

[1379] Destructor(int fd, char *cp)

[1380] {

[1381] ste_t *ste_params=NULL;

[1382] sd_t *sd=NULL;

[1383] int rc=-1;

[1384] Hash_position_t *hp=NULL;

[1385] Hash_bucket_t *hb=NULL;

[1386] if(preprocess_request(cp, &sd, &ste_params, 1) == -1) return(-1);

[1387] if(!sd || ! ste_params || ! sd->ste_table) return(-1);

[1388] hp=hashscan(sd->ste_table,0);

[1389] while((hb=hashnext(hp)) != NULL) {
a. char *ste_name=hashname(hb);
b. ste_t *ste;
c. ste = (ste_t *) hashget (sd->ste_table, ste_name);
d. if(fd == 0) fd=2;
e. if(compare_ste(fd, ste, ste_params) == 1) {
f. hashlook(sd->ste_table,ste_name,HASH_DELETE,0);
g. ste_f.destructor(ste);
h. rc=0;
i. }

[1390] }

```
[1391]  return(rc);
[1392]  }
[1393]  static int
[1394]  SetSystemWideSD()
[1395]  {
[1396]  char buffer[4096];
[1397]  char *s;
[1398]  pid_t pid;
[1399]  struct hostent *sp=NULL;
[1400]  struct  utsname uts;
[1401]  sd_t  *sd=NULL;
[1402]  ste_t *ste=NULL;
[1403]  if( ! sd_base.sd_table) sd_base.sd_table= hashalloc(NULL,HASH_name,"BASETDS",0);
[1404]  if( ! sd_base.command_table)
[1405]  sd_base.command_table= hashalloc(NULL,HASH_name,"BASECOMMANDS",0);
[1406]  pid=getpid();
[1407]  memset( &uts, '\0', sizeof(struct utsname));
[1408]  uname( &uts );
[1409]  sp = gethostbyname(uts.nodename);
[1410]  if( sp && sp->h_addr)  s = make_string( (char *)sp->h_addr);
[1411]  else s=make_string((char *)"N/A");
[1412]  memset(buffer,'\0',4096);
[1413]  sfsprintf(buffer,4096,"type=sd sd=default");
[1414]  if( sd_f.constructor(buffer) == NULL )
        a.  fprintf(stderr,"failed to build the default table. buffer=[%s]\n",buffer);
[1415]  sd=(sd_t *)hashget(sd_base.sd_table,"default");
[1416]  if(!sd) { fprintf(stderr,"initialization of default SD failed\n");   exit(1); }
[1417]  sfsprintf(buffer,4096,"type=sd tds=default name=tds sd=default sdaddr=%x pid=%d sysname=%s sysnode=%s
sysrelease=%s sysversion=%s sysmachine=%s ipaddr=%s",
        a.  sd,pid,uts.sysname,uts.nodename,uts.release,uts.version,uts.machine,s);
[1418]  ste=NULL;
[1419]  sd=NULL;
[1420]  preprocess_request(buffer, &sd, &ste, 1);
[1421]  if( !sd || !ste )
[1422]  fprintf(stderr,"ERROR: internal error when creating the first service type entry for the default service directory\n");
[1423]  ste->sd=sd;
```

```
[1424]  hashput(sd->ste_table,ste->ste_id.id,ste);
[1425]  sd->command_table = hashalloc(NULL,HASH_name,"COMMANDS",0);
[1426]  RegisterCommand(sd->command_table,"register","register",NULL,sd_f.register_ste);
[1427]  RegisterCommand(sd->command_table,"create", "create",  NULL,sd_f.constructor);
[1428]  RegisterCommand(sd->command_table,"query", "query",   NULL,sd_f.query);
[1429]  RegisterCommand(sd->command_table,"delete", "delete",  NULL,sd_f.destructor);
[1430]  return(0);
[1431]  }
[1432]  static int isize=0;
[1433]  static char *input;
[1434]  static int      (*DefaultDescramble)(int, char *);
[1435]  int
[1436]  init_config_file(void **handle)
[1437]  {
[1438]  char buffer[2048];
[1439]  int fd=open("tds.config",O_RDONLY);
[1440]  if( fd < 0 ) return(-1);
[1441]  while( readline(fd,buffer,2048) > 0 ) {
a.  char  *command=NULL;
b.  char  *name=NULL;
c.  char  *location=NULL;
d.  char *cp=buffer;
e.  char *n=NULL, *v=NULL;
f.  while( *cp ) {
g.  cp=getnvpair(cp,&n, &v);
h.  if( ! n || !v ) break;
i.  if( strcmp(n,"command")==0) command=v;
j.  else if(strcmp(n,"location")==0) location=v;
k.  else if(strcmp(n,"name")==0) name=v;
l.  }
m.  if( command && strcmp(command,"SetSystemWideSD")==0) {
n.  if(name && location ) {
        i.  void *h;
        ii.  int (*f)()=NULL;
        iii.  h=dlopen(name,RTLD_LAZY);
        iv.  if( h ) f=dlsym(h,command);
        v.  if( f && handle ) *handle = f;
o.  }
p.  }
q.  if( command && strcmp(command,"DefaultDescramble")==0) {
r.  if(name && location ) {
        i.  void *h;
        ii.  int (*f)()=NULL;
```

```
                iii.  h=dlopen(name,RTLD_LAZY);
                iv.   if( h ) f=dlsym(h,command);
                 v.   if( f && handle ) DefaultDescramble = f;
          s.    }
          t.    }
[1442]          }
[1443]          return(0);
[1444]          }

[1445]          int tds(int fd, void *args)
[1446]          {
[1447]          char        *command_name=NULL;
[1448]          char        *sd_id=NULL;
[1449]          char        buffer[PIPE_BUF];
[1450]          char        *bcp=NULL;
[1451]          int         ni=0;
[1452]          int         rc=-1;
[1453]          char        response[100];
[1454]          sd_t        *sd=NULL;
[1455]          command_t       *command=NULL;
[1456]          command_t       *authentication=NULL;
[1457]          int         (*setdefault)()=SetSystemWideSD;
[1458]          char        *cp=NULL;
[1459]          Hash_table_t *command_table=NULL;
[1460]          Hash_table_t *default_command_table=NULL;
[1461]          if( ! sd_base.initialized ) {
          a.    init_config_file( (void **) &setdefault );
          b.    if( setdefault ) {
          c.    if( setdefault() == 0 )  sd_base.initialized=1;
          d.    else setdefault=NULL;
          e.    }
          f.    if( ! setdefault )
[1462]          { fprintf(stderr,"initialization sequence failed.\n"); return(-1); }
[1463]          }
[1464]          wait_read(fd,30);
[1465]          ni=peek(fd);
[1466]          if(ni<0)    goto response;
[1467]          if(! isize )
[1468]          { input=malloc(2048); isize=2048; }
```

```
[1469]      memset(input,'\0',isize);
[1470]      ni=readline(fd,input,isize);
[1471]      if( ni < 0 ) goto response;
[1472]      cp=input;
[1473]      if(!cp) goto response;
[1474]      if( DefaultDescramble)      DefaultDescramble(fd,cp);
[1475]      memset(buffer,'\0',PIPE_BUF);
[1476]      memcpy(buffer,cp, strlen(cp) < PIPE_BUF ? strlen(cp) : PIPE_BUF - 1);
[1477]      bcp  = buffer;
[1478]      while(*bcp && isspace(*bcp)) ++bcp;
[1479]      while(*bcp) {
        a.  char *n=NULL, *v=NULL;
        b.  bcp=(char *)getnvpair(bcp,&n,&v);
        c.  if(!n || !v ) break;
        d.  if( strcasecmp(n,"command")==0) command_name=v;
        e.  else if(strcasecmp(n,"sd")==0) sd_id=v;
        f.  break;
[1480]      }
[1481]      sd=(sd_t *)hashget(sd_base.sd_table,sd_id ? sd_id : "default");
[1482]      if(!sd)      goto response;
[1483]      command_table = sd->command_table;
[1484]      sd=(sd_t *)hashget(sd_base.sd_table,"default");
[1485]      if(!sd)      goto response;
[1486]      default_command_table = sd->command_table;
[1487]      if( !sd || !sd->command_table)    goto response;
[1488]      if( command_name) {
        a.  if(command_table) {
        b.  command=(command_t *)hashget(command_table,command_name);
        c.  authentication=(command_t *)hashget(command_table,"authenticate");
        d.  }
        e.  if(!command)    command=(command_t *)hashget(default_command_table,command_name);
        f.  if(!authentication ||!authentication->function)
        g.  authentication=(command_t *)hashget(default_command_table,"authenticate");
        h.  if(authentication && authentication->function)
        i.  if(authentication->function(fd,cp)) goto response;
        j.  if( command && command->function )
        k.  rc=command->function(fd,cp);
[1489]      }
[1490]      response:
[1491]      sfsprintf(response,100,"status=%d\n",rc != -1 ? 0 : -1);
```

```
[1492]  if( write(fd,response,strlen(response)) != strlen(response))
[1493]  fprintf(stderr,"write response failed errno=%d\n", errno);
[1494]  return(0);
[1495]  }
[1496]  sd_f_t sd_f = { Constructor, Destructor, Query, Register };
```

[1497] Program Listing 4.2 - Thread Directory Service - ste.c

```
[1498]  #include <ast.h>
[1499]  #include <stdio.h>
[1500]  #include <sfio.h>
[1501]  #include <hash.h>
[1502]  #include <memory.h>
[1503]  #include "tds2.h"
[1504]  static char ics_buffer[100];
[1505]  static int counter;
[1506]  static char *make_string(char *m)
[1507]  {
[1508]  char *cp;
[1509]  if(!m) return(NULL);
[1510]  if( !(cp=malloc(strlen(m)+1)) )
[1511]  { fprintf(stderr,"out of resources on size: [%d]\n",strlen(m)+1); return(NULL); }
[1512]  strcpy(cp,m);
[1513]  return(cp);
[1514]  }
[1515]  static ste_t * Constructor(void)
[1516]  {
[1517]  ste_t *s=malloc(sizeof(ste_t));
[1518]  if(s) memset(s,'\0',sizeof(ste_t));
[1519]  return(s);
[1520]  }
[1521]  static ic_t * AddIC(ste_t *ste, ic_t *ic, long private)
[1522]  {
[1523]  if( ! ste->ic_table ) ste->ic_table = hashalloc(NULL,HASH_name,"ICTABLE",0);
[1524]  ic->ste = ste;
[1525]  ic->status = private;
```

```
[1526]      hashput(ste->ic_table, ic->name, ic);
[1527]      return(0);
[1528]      }
[1529]      static int Destructor(ste_t *ste)
[1530]      {
[1531]      Hash_position_t *hp=NULL;
[1532]      Hash_bucket_t  *hb=NULL;
[1533]      if( !ste) return(-1);
[1534]      hp=hashscan(ste->ic_table,0);
[1535]      while( (hb=hashnext(hp)) != NULL) {
              a. char *ic_name;
              b. ic_t  *ic;
              c. ic_name=hashname(hb);
              d. ic=(ic_t *)hashget(ste->ic_table,ic_name);
              e. hashlook(ste->ic_table,ic_name,HASH_DELETE,0);
[1536]      }
[1537]      hashdone(hp);
[1538]      hashfree(ste->ic_table);
[1539]      if(ste->ste_id.id)      free(ste->ste_id.id);
[1540]      if(ste->ste_id.name)    free(ste->ste_id.name);
[1541]      if(ste->ste_id.location) free(ste->ste_id.location);
[1542]      if(ste->ste_id.physical) free(ste->ste_id.physical);
[1543]      free(ste);
[1544]      return(0);
[1545]      }
[1546]      ste_f_t ste_f = { AddIC, Constructor, Destructor };
```

[1547] Program Listing 4.3 - Thread Directory Service - log.c

```
[1548]      #include <ast.h>
[1549]      #include <sfio.h>
[1550]      #include <stdio.h>
[1551]      #include <fcntl.h>
[1552]      #include "log.h"
[1553]      static int index;
[1554]      static int LogMessage(char *message, int code)
[1555]      {
[1556]      int i;
```

```
[1557]  FILE *fp=fopen("/usr/tmp/log","a");
[1558]  for(i=0; i < index; ++i) fprintf(fp," ");
[1559]  fprintf(fp,"%s ",message);
[1560]  if( code ) fprintf(fp,"%d", code);
[1561]  fprintf(fp,"\n");
[1562]  fclose(fp);
[1563]  }
[1564]  static int LogEnter(char *message, int code)
[1565]  {
[1566]  char gbuff[4096];
[1567]  sfsprintf(gbuff,4096,">%s",message);
[1568]  LogMessage(gbuff,code);
[1569]  index+=3;
[1570]  return(0);
[1571]  }
[1572]  static int LogExit(char *message, int code)
[1573]  {
[1574]  char gbuff[4096];
[1575]  index-=3;
[1576]  sfsprintf(gbuff,4096,"<%s",message);
[1577]  LogMessage(gbuff,code);
[1578]  return(0);
[1579]  }
[1580]  log_f_t log_f =  { LogMessage, LogEnter, LogExit };
```

[1581] Program Listing 4.4 Thread Directory Service - ic.c

```
[1582]  #include <ast.h>
[1583]  #include <stdio.h>
[1584]  #include <sfio.h>
[1585]  #include <hash.h>
[1586]  #include <memory.h>
[1587]  #include "tds2.h"
[1588]  static char *make_string(char *m)
[1589]  {
[1590]  char *cp;
```

```
[1591]  if(!m) return(NULL);
[1592]  if( !(cp=malloc(strlen(m)+1)) )
[1593]  { fprintf(stderr,"out of resourcs on size: [%d]\n",strlen(m)+1); return(NULL); }
[1594]  strcpy(cp,m);
[1595]  return(cp);
[1596]  }
[1597]  static ic_t *
[1598]  Allocate(void)
[1599]  {
[1600]  ic_t *ic;
[1601]  ic=malloc(sizeof(ic_t));
[1602]  if(ic) memset(ic,'\0',sizeof(ic_t));
[1603]  return(ic);
[1604]  }
[1605]  static ic_t *
[1606]  Constructor(char *name, char *value)
[1607]  {
[1608]  ic_t *ic = Allocate();
[1609]  if(!ic) return(NULL);
[1610]  ic->name=make_string(name);
[1611]  ic->value=make_string(value);
[1612]  return(ic);
[1613]  }
[1614]  ic_f_t ic_f = { Constructor };
```

[1615] Program Listing 4.5 thread directory service - set_blocking.c

```
[1616]  #include <stdio.h>
[1617]  #include <fcntl.h>
[1618]  int set_blocking( int fd )
[1619]  {
[1620]  int flags;
[1621]  if((flags=fcntl(fd, F_GETFL,0)) < 0) return(-1);
[1622]  flags &= ~O_NONBLOCK;
[1623]  if(fcntl(fd,F_SETFL,flags) < 0 ) return(-1);
[1624]  return(0);
```

[1625] }

[1626] Program Listing 4.6 thread directory service - set_nonblocking.c

```
[1627] #include <stdio.h>
[1628] #include <fcntl.h>
[1629] int set_nonblocking( int fd )
[1630] {
[1631] int flags;
[1632] if((flags=fcntl(fd, F_GETFL,0)) < 0) return(-1);
[1633] flags |= O_NONBLOCK;
[1634] if(fcntl(fd,F_SETFL,flags) < 0 ) return(-1);
[1635] return(0);
[1636] }
```

[1637] Program Listing 4.7 thread directory service - Makefile

```
[1638] :PACKAGE: ast
[1639] CCFLAGS  = -D_BLD_ast $(CC.DLL) -Wall
[1640] tds :LIBRARY: ic.c ste.c getnvpair.c readline.c peek.c peek_c.c wait_read.c set_blocking.c set_nonblocking.c log.c tds3.c
```

[1641] Program Listing 5.0 fopen service - fopen.c

```
[1642] #include <stdio.h>
[1643] int fopen_service(int fd, char *argv[])
[1644] {
[1645] int fd_in=0;
[1646] int fd_out=1;
[1647] int n;
[1648] FILE *fp;
[1649] char *p="what?\n";
[1650] char buffer[2048];
[1651] char mode[10];
[1652] if( argv == NULL ) { fd_in=fd; fd_out=fd;}
[1653] memset(buffer,'\0',2048);
[1654] write(fd_out,p,strlen(p));
[1655] n=readline(fd_in,buffer,2048);
```

```
[1656]  if( n > 0 ) buffer[n-1]='\0';
[1657]  memset(mode,'\0',10);
[1658]  write(fd_out,"mode?\n",6);
[1659]  n=readline(fd_in,mode,10);
[1660]  if( n > 0 ) mode[n-1]='\0';
[1661]  fp=fopen(buffer,mode);
[1662]  sprintf(buffer,"%x",fp);
[1663]  write(fd_out,buffer,strlen(buffer));
[1664]  return(0);
[1665]  }
```

[1666] Program Listing 6.0 fscanf service - fscanf.c

```
[1667]  #include <stdio.h>
[1668]  int fscanf_service(int fd, char *argv[])
[1669]  {
[1670]  int fd_in=0;
[1671]  int fd_out=1;
[1672]  int n;
[1673]  FILE *fp;
[1674]  char *p="with what?\n";
[1675]  char buffer[2048];
[1676]  char mode[10];
[1677]  if( argv == NULL ) { fd_in=fd; fd_out=fd;}
[1678]  memset(buffer,'\0',2048);
[1679]  write(fd_out,p,strlen(p));
[1680]  n=readline(fd_in,buffer,2048);
[1681]  if( n > 0 ) buffer[n-1]='\0';
[1682]  sscanf(buffer,"%x", &fp);
[1683]  memset(mode,'\0',10);
[1684]  write(fd_out,"for?\n",6);
[1685]  n=readline(fd_in,mode,10);
[1686]  if( n > 0 ) mode[n-1]='\0';
[1687]  if( strcmp(mode,"s")==0) {
a.      memset(buffer,'\0',2048);
b.      fscanf(fp,"%s",buffer);
c.      write(fd_out,buffer,strlen(buffer));
```

```
}
return(0);
}
```

Program Listing 7.0 fclose service - fclose.c

```
#include <stdio.h>
int
fclose_service(int fd, char *argv[])
{
int fd_in=0;
int fd_out=1;
int n;
FILE *fp;
char *p="with what?\n";
char buffer[2048];
char mode[10];
if( argv == NULL )
{ fd_in=fd; fd_out=fd; }
memset(buffer,'\0',2048);
n=readline(fd_in,buffer,2048);
if( n > 0 ) buffer[n-1]='\0';
sscanf(buffer,"%x", &fp);
fclose(fp);
return(0);
}
```

Program Listing 8.0 caps service caps.c

```
#include <stdio.h>
#include <dlfcn.h>
#include <malloc.h>

typedef
struct _srvc {
char *location;
char *name;
```

```
[1720]  char *primitive;
[1721]  char *physical;
[1722]  void  *handle;
[1723]  int  (*function)();
[1724]  struct _srvc *last;
[1725]  struct _srvc *next;
[1726]  } srvc_t;
[1727]  typedef
[1728]  struct _srvcbase {
[1729]  srvc_t      *head;
[1730]  srvc_t      *tail;
[1731]  } srvcbase_t;
[1732]  srvcbase_t srvcbase;
[1733]  static srvc_t *
[1734]  Constructor(void)
[1735]  {
[1736]  srvc_t *srvc = malloc(sizeof(srvc_t));
[1737]  srvc_t *sp;
[1738]  memset(srvc,'\0',sizeof(srvc_t));
[1739]  if( ! srvcbase.head ) { srvcbase.head = srvc; srvcbase.tail = srvc; }
[1740]  else { srvc->last = srvcbase.tail; srvcbase.tail->next = srvc; srvcbase.tail=srvc; }
[1741]  return(srvc);
[1742]  }
[1743]  static int
[1744]  Destructor(srvc_t *srvc)
[1745]  {
[1746]  srvc_t *sp;
[1747]  if(!srvc) return(1);
[1748]  if( srvcbase.head == srvc ) srvcbase.head = srvc->next;
[1749]  if( srvcbase.tail == srvc ) srvcbase.tail = srvc->last;
[1750]  if( srvc->last ) srvc->last->next = srvc->next;
[1751]  if( srvc->next ) srvc->next->last = srvc->last;
[1752]  if( srvc->primitive) free(srvc->primitive);
[1753]  if( srvc->physical)  free(srvc->physical);
[1754]  if( srvc->location)  free(srvc->location);
[1755]  if( srvc->name)  free(srvc->name);
```

```
[1756]  if(srvc->handle) dlclose(srvc->handle);
[1757]  free(srvc);
[1758]  return(0);
[1759]  }
[1760]  int caps_service(int fd, char *argv[])
[1761]  {
[1762]  int fd_in=0;
[1763]  int fd_out=1;
[1764]  int r;
[1765]  FILE *fp;
[1766]  char *cp;
[1767]  char *p="what?\n";
[1768]  char buffer[2048];
[1769]  char criteria[2048];
[1770]  char mode[10];
[1771]  char *command=NULL, *arg=NULL;
[1772]  char *n=NULL, *v=NULL;
[1773]  if( argv == NULL ) { fd_in=fd; fd_out=fd;}
[1774]  write(fd_out,p,strlen(p));
[1775]  memset(buffer,'\0',2048);
[1776]  r=readline(fd_in,buffer,2048);
[1777]  if( r > 0 ) { if( buffer[r-1]=='\n') buffer[r-1]='\0'; }
[1778]  command=buffer;
[1779]  while( isspace(*command)) ++command;
[1780]  arg=command;
[1781]  while( *arg && !isspace(*arg)) ++arg;
[1782]  if( *arg ) { *arg = '\0'; ++arg; }
[1783]  if( strcmp(command,"EXECUTE")==0) {
a.      char *name=NULL,*location=NULL,*primitive=NULL,*physical=NULL;
b.      srvc_t *srvc;
c.      char  **args=NULL;
d.      int   *argc=NULL;
e.      tds_query_p(arg,buffer,2048);
f.      tds_get_nlpp(buffer, &name, &location, &primitive, &physical, NULL,NULL);
g.      for(srvc=srvcbase.head; srvc; srvc=srvc->next) {
h.      int m=0;
i.      if( name ) if( srvc->name && strcmp(srvc->name,name)!=0)
            i.  continue;
j.      if( location ) if( srvc->location && strcmp(srvc->location,location)!=0)
```

```
                i.  continue;
        k.  if( physical ) if( srvc->physical && strcmp(srvc->physical,physical)!=0)
                i.  continue;
        l.  if( primitive) if( srvc->primitive && strcmp(srvc->primitive,primitive)!=0)
                i.  continue;
        m.  break;
        n.  }
[1784]      if(argc) fprintf(stderr,"argc=%d\n", *argc);
        a.  if( srvc && srvc->function ) {
        b.  if( argc ) return( (*srvc->function)(*argc,args) );
        c.  else return( (*srvc->function)(fd,argv) );
        d.  }
[1785]      }
[1786]      if( strcmp(command,"LOCATE")==0)
[1787]      {
        a.  int n;
        b.  char *name=NULL,*location=NULL,*primitive=NULL,*physical=NULL;
        c.  tds_query_p(arg,buffer,2048);
        d.  tds_get_nlpp(buffer, &name, &location, &primitive, &physical);
        e.  memset(buffer,'\0',2048);
        f.  sprintf(buffer,"%s %s %s %s\n", name  ? name : "",
        g.  location ? location : "", primitive ? primitive : "",
        h.  physical  ? physical : "" );
        i.  write(fd_out,buffer,strlen(buffer));
        j.  write(fd_out,buffer,"\n",1);
        k.  if( name )   free(name);
        l.  if(location)  free(location);
        m.  if(primitive) free(primitive);
        n.  if(physical)  free(physical);
[1788]      }
[1789]      if(strcmp(command,"LOAD") == 0 )
[1790]      {
        a.  int n;
        b.  char *name=NULL,*location=NULL,*primitive=NULL,*physical=NULL;
        c.  srvc_t *srvc = Constructor();
        d.  tds_query_p(arg,buffer,2048);
        e.  tds_get_nlpp(buffer, &name, &location, &primitive, &physical);
        f.  sprintf(buffer,"%s %s %s %s\n", name  ? name : "",
        g.  location ? location : "",primitive ? primitive : "",physical  ? physical : "" );
        h.  srvc->name    = name;
        i.  srvc->location = location;
        j.  srvc->primitive= primitive;
        k.  srvc->physical = physical;
        l.  srvc->handle=dlopen(srvc->location, RTLD_LAZY);
        m.  if(srvc->handle) srvc->function=dlsym(srvc->handle, srvc->name);
[1791]      }
```

```
[1792]      if(strcmp(command,"UNLOAD")==0)
[1793]      {
a.          char *name=NULL,*location=NULL,*primitive=NULL,*physical=NULL;
b.          srvc_t *srvc;
c.          char  **args=NULL;
d.          int   *argc=NULL;
e.          tds_query_p(arg,buffer,2048);
f.          tds_get_nlpp(buffer, &name, &location, &primitive, &physical, &argc, &args);
g.          for(srvc=srvcbase.head; srvc; srvc=srvc->next) {
h.          int m=0;
i.          if( name ) if( srvc->name && strcmp(srvc->name,name)!=0)
                i.  continue;
j.          if( location ) if( srvc->location && strcmp(srvc->location,location)!=0)
                i.  continue;
k.          if( physical ) if( srvc->physical && strcmp(srvc->physical,physical)!=0)
                i.  continue;
l.          if( primitive) if( srvc->primitive && strcmp(srvc->primitive,primitive)!=0)
                i.  continue;
m.          break;
n.          }
o.          if(srvc)    Destructor(srvc);
[1794]      }
[1795]      return(0);
[1796]      }
```

[1797] Program Listing 9.1 generic front end loader service gfel.c

```
[1798]  #include <stdio.h>
[1799]  #include <dlfcn.h>
[1800]  #include <errno.h>
[1801]  #include <malloc.h>
[1802]  #include <limits.h>
[1803]  #include "log.h"
[1804]  int get_tds_connection_info(char **, char **);
[1805]  int gfel(int argc, char *argv[], int inputdescriptor, int outputdescriptor)
[1806]  {
[1807]  void  *handle;
[1808]  int   (*function)(int, char **);
[1809]  void  *libthandle;
[1810]  int   (*tcp_accept)(int, char *, char *, char *, char *);
[1811]  char  buffer[4096];
[1812]  char  *service_name=NULL;
```

```
[1813]      char *service_location=NULL;
[1814]      char *service_primitive=NULL;
[1815]      char *service_connectivity=NULL;
[1816]      char *computer=NULL;
[1817]      char *service=NULL;
[1818]      char *service_type=NULL;
[1819]      char *name=NULL;
[1820]      char *value=NULL;
[1821]      int   index;
[1822]      int   pipefds[2];
[1823]      if(argc == 1 ) {
        a.  fprintf(stderr,"gl2 name=_name_ location=_location_ primitive=_primitive_ physical=_physical_\n");
        b.  exit(0);
[1824]      }
[1825]      service_name = service_location = NULL;
[1826]      for(index=1; index < argc; ++index) {
        a.  name=value=NULL;
        b.  getnvpair(argv[index], &name, &value);
        c.  if(name) {
        d.  if(strcmp(name,"name")==0) service_name=value;
        e.  else if(strcmp(name,"location")==0) service_location=value;
        f.  else if(strcmp(name,"primitive")==0) service_primitive=value;
        g.  else if(strcmp(name,"physical")==0) service_connectivity=value;
        h.  else break;
        i.  }
        j.  else break;
[1827]      }
[1828]      argv=&argv[index];
[1829]      argc-=index;
[1830]      // if service connectivity is specified, then we assume we
[1831]      // are a service provider.  We allow two portions in the
[1832]      // connectivity specification.  the first is the host, and the
[1833]      // second is the service.  This allows us to use things like:
[1834]      //      127.0.0.1:999
[1835]      //      /local:/tmp/unix_domain_socket
[1836]      //      /fifo:/tmp/named_pipe
[1837]      //      /mmap:/path/filename
[1838]      //      /phone:609-924-07305
[1839]      //      /email:cjn@gtlinc.com
```

[1840] // /pager:609-924-7305

[1841] // /fax:609-924-7306

[1842] if(service_connectivity) {

a. service=computer=service_connectivity;

b. while(*service && *service != ':') ++service;

c. if(*service) { *service='\0'; ++service; }

[1843] }

[1844] if(computer && service && service_name && strcmp(service_primitive,"INET")==0)

[1845] {

a. char *tds_computer=NULL;

b. char *tds_service=NULL;

c. libthandle = dlopen("../tcs/libtcs.so.1.0",RTLD_LAZY);

d. if(!libthandle) {

e. fprintf(stderr,"configuration error on TCS\n");

f. fprintf(stderr,"error condition: %s\n",dlerror());

g. exit(1);

h. }

i. tcp_accept=dlsym(libthandle,"tcp_accept2");

j. if(!tcp_accept) {

k. fprintf(stderr,"TCS configuration error for supporting routines.\n");

l. exit(1);

m. }

n. if(strcasecmp(service_name,"tds")==0) {

o. void *handle;

p. int (*tds)(int, char *);

q. FILE *fp=fopen("/tmp/tds","a");

r. fprintf(fp,"inet %s %s\n", computer, service);

s. fclose(fp);

t. }

u. else {

v. char gbuff[PIPE_BUF];

w. int rsize=2048;

x. get_tds_connection_info(&tds_computer, &tds_service);

y. if(tds_computer && tds_service) {

i. sfsprintf(gbuff,2048,"command=register name=%s location=%s physical=%s:%s primitive=%s pid=%d ppid=%d\nQUIT\n", service_name, service_location, computer, service, service_primitive, getpid(), getppid());

ii. sendreceive(tds_computer,tds_service,gbuff,strlen(gbuff),buffer,&rsize);

z. }

aa. else fprintf(stderr,"failed to find TDS to register.\n");

bb. }

cc. (*tcp_accept)(argc, computer, service, service_location, service_name);

dd. exit(0);

[1846] }

[1847] while(1) {

a. handle = dlopen(service_location == NULL ? argv[1] : service_location, RTLD_LAZY);

b. if(! handle) break;

c. if(! service_name) service_name=argv[2];

d. function = dlsym(handle, service_name);
e. if(! function) { dlclose(handle); break; }
f. return ((*function) (argc, argv));

[1848] }

[1849] if(service_name && ! service_location) {
a. char *tds_computer=NULL;
b. char *tds_service=NULL;
c. get_tds_connection_info(&tds_computer, &tds_service);
d. if(tds_computer && tds_service) {
e. char gbuff[4096];
f. char *bp;
g. int rsize=4096;
h. memset(buffer,'\0',rsize);
i. sprintf(gbuff,"command=query name=%s\n", service_name);
j. if(sendreceive(tds_computer,tds_service,gbuff,2048,buffer,&rsize) <=0)

[1850] { fprintf(stderr,"failed to locate service\n"); return(-1); }
a. bp=buffer;
b. while(bp) {
i. name=NULL;
ii. value=NULL;
iii. bp=(char *)getnvpair(bp, &name, &value);
iv. if(! name || ! value) break;
v. if(strcmp(name,"location")==0) {
vi. service_location=malloc(strlen(value)+1);
vii. strcpy(service_location,value);
viii. }
ix. else if(strcmp(name,"primitive")==0) {
x. service_primitive=malloc(strlen(value)+1);
xi. strcpy(service_primitive,value);
xii. }
xiii. else if(strcmp(name,"physical")==0) {
xiv. service_connectivity=malloc(strlen(value)+1);
xv. strcpy(service_connectivity,value);
xvi. }
c. }
d. if(service_connectivity) {
i. service=computer=service_connectivity;
ii. while(*service && *service != ':') ++service;
iii. if(*service) { *service='\0'; ++service; }
e. }
f. client_gl2(computer, service, inputdescriptor, outputdescriptor);
g. }

[1851] }

[1852] exit(0);

[1853] }

[1854] int get_tds_connection_info(char **computer, char **service)

[1855] {

```
[1856]      char buffer[2048];
[1857]      FILE *fp=fopen("/tmp/tds","r");
[1858]      while( fscanf(fp,"%s",buffer) == 1 ) {
        a.  if(strcmp(buffer,"inet")==0) {
        b.  memset(buffer,'\0',2048);
        c.  fscanf(fp,"%s",buffer);
        d.  *computer=malloc(strlen(buffer)+1);
        e.  strcpy(*computer,buffer);
        f.  memset(buffer,'\0',2048);
        g.  fscanf(fp,"%s",buffer);
        h.  *service=malloc(strlen(buffer)+1);
        i.  strcpy(*service,buffer);
        j.  break;
        k.  }
[1859]      }
[1860]      fclose(fp);
[1861]      }
```

[1862] Program Listing 9.2 generic front end loader service client_gl.c

```
[1863]  #include "services.h"
[1864]  #define MAXSOCKADDR  2048
[1865]  int client_gl(char *argv1, char *argv2, int fd_in, int fd_out)
[1866]  {
[1867]  int sockfd=-1;
[1868]  int n;
[1869]  char recvline[2048];
[1870]  socklen_t  len;
[1871]  struct sockaddr *sa;
[1872]  int   fd_set1[2];
[1873]  int   fd_set2[2];
[1874]  void  *libhandle;
[1875]  void  *tcplibhandle;
[1876]  int   (*function)(int *, int *);
[1877]  int   (*gamixconnect)(char *, char *);
[1878]  if( (tcplibhandle = dlopen("../tcs/libtcs.so.1.0",RTLD_LAZY)) == NULL) {
        a.  fprintf(stderr,"dlopen libservices.so failed errno=%d\n", errno);
        b.  fprintf(stderr,"error condition [%s]\n", dlerror());
[1879]  }
[1880]  gamixconnect=dlsym(tcplibhandle,"tcp_connect");
```

```
[1881]      if(!gamixconnect) {
        a.  fprintf(stderr,"tcp_connect missing\n");
        b.  fprintf(stderr,"error condition [%s]\n", dlerror());
[1882]      }
[1883]      if( (function  = dlsym(tcplibhandle, "talk2")) == NULL) {
        a.  fprintf(stderr,"dlsym of function [%s] failed errno=%d\n", "talk2", errno);
        b.  fprintf(stderr,"error condition [%s]\n", dlerror());
        c.  exit(1);
[1884]      }
[1885]      memset(recvline,'\0',2048);
[1886]      if(gamixconnect && ( (sockfd=(*gamixconnect)(argv1,argv2)) == -1 )) exit(1);
[1887]      else exit(1);
[1888]      sa=malloc(MAXSOCKADDR);
[1889]      len=MAXSOCKADDR;
[1890]      getpeername(sockfd,sa,&len);
[1891]      // setup the connector
[1892]      fd_set1[0]=fd_in;
[1893]      fd_set1[1]=dup(sockfd);
[1894]      fd_set2[0]=dup(sockfd);
[1895]      fd_set2[1]=fd_out;
[1896]      close(sockfd);
[1897]      (*function)(fd_set1, fd_set2);
[1898]      // now shutdown, and then close, the connector
[1899]      shutdown(fd_set1[1],2);
[1900]      shutdown(fd_set2[0],2);
[1901]      close(fd_set1[1]);
[1902]      close(fd_set2[0]);
[1903]      return(0);
[1904]      }
```

[1905] Program Listing 9.3 generic front end loader service client_gl2.c

```
[1906]      #include "services.h"
[1907]      #include "log.h"
[1908]      #define MAXSOCKADDR  2048
[1909]      int
[1910]      client_gl2(char *argv1, char *argv2, int fd_in, int fd_out)
[1911]      {
```

```
[1912]      int sockfd=-1;
[1913]      int n;
[1914]      char recvline[2048];
[1915]      socklen_t  len;
[1916]      struct sockaddr *sa;
[1917]      int   fd_set1[2];
[1918]      int   fd_set2[2];
[1919]      void  *libhandle;
[1920]      void  *tcplibhandle;
[1921]      int   (*function)(int *, int *);
[1922]      int   (*gamixconnect)(char *, char *);
[1923]      if( (tcplibhandle = dlopen("../tcs/libtcs.so.1.0",RTLD_LAZY)) == NULL) {
        a.  fprintf(stderr,"dlopen ../tcs/libtcs.so failed errno=%d\n", errno);
        b.  fprintf(stderr,"error condition [%s]\n", dlerror());
[1924]      }
[1925]      gamixconnect=dlsym(tcplibhandle,"tcp_connect");
[1926]      if(!gamixconnect) {
        a.  fprintf(stderr,"tcp_connect missing\n");
        b.  fprintf(stderr,"error condition [%s]\n", dlerror());
[1927]      }
[1928]      if( (function  = dlsym(tcplibhandle, "talk2")) == NULL) exit(1);
[1929]      memset(recvline,'\0',2048);
[1930]      if(gamixconnect && ( (sockfd=(*gamixconnect)(argv1,argv2)) == -1 )) exit(1);
[1931]      else exit(1);
[1932]      sa=malloc(MAXSOCKADDR);
[1933]      len=MAXSOCKADDR;
[1934]      getpeername(sockfd,sa,&len);
[1935]      // setup the connector
[1936]      fd_set1[0]=fd_in;
[1937]      fd_set1[1]=dup(sockfd);
[1938]      fd_set2[0]=dup(sockfd);
[1939]      fd_set2[1]=fd_out;
[1940]      close(sockfd);
[1941]      (*function)(fd_set1, fd_set2);
[1942]      // now shutdown, and then close, the connector
[1943]      if( shutdown(fd_set1[1],2) != 0 )
[1944]      fprintf(stderr,"client_gl2: shutdown failed error=%d\n",errno);
```

```
[1945]  if( close(fd_set1[1]) ) fprintf(stderr,"client_gl2: close fd_set1[1] failed error=%d\n", errno);
[1946]  if( close(fd_set2[0]) ) fprintf(stderr,"client_gl2: close fd_set2[0] failed error=%d\n", errno);
[1947]  return(0);
[1948]  }
```

[1949] Program Listing 9.4 generic front end loader service gl3.c

```
[1950]  #include <stdio.h>
[1951]  #include <dlfcn.h>
[1952]  #include <errno.h>
[1953]  #include <malloc.h>
[1954]  #include "log.h"
[1955]  int get_tds_connection_info(char **, char **);
[1956]  int gfel(int argc, char *argv[])
[1957]  {
[1958]  void  *handle;
[1959]  int   (*function)(int, char **);
[1960]  void  *libthandle;
[1961]  int   (*tcp_accept)(int, char *, char *, char *, char *);
[1962]  char  buffer[2048];
[1963]  char  *service_name=NULL;
[1964]  char  *service_location=NULL;
[1965]  char  *service_primitive=NULL;
[1966]  char  *service_connectivity=NULL;
[1967]  char  *computer=NULL;
[1968]  char  *service=NULL;
[1969]  char  *service_type=NULL;
[1970]  char  *name=NULL;
[1971]  char  *value=NULL;
[1972]  int   index;
[1973]  int   pipefds[2];
[1974]  if(argc == 1 ) {
a.      fprintf(stderr,"gl2 name=_name_ location=_location_ primitive=_primitive_ physical=_physical_\n");
b.      exit(0);
[1975]  }
[1976]  service_name = service_location = NULL;
[1977]  for(index=1; index < argc; ++index) {
```

a. name=value=NULL;
b. getnvpair(argv[index], &name, &value);
c. if(name) {
d. if(strcmp(name,"name")==0) service_name=value;
e. else if(strcmp(name,"location")==0) service_location=value;
f. else if(strcmp(name,"primitive")==0) service_primitive=value;
g. else if(strcmp(name,"physical")==0) service_connectivity=value;
h. else break;
i. }
j. else break;

[1978] }

[1979] argv=&argv[index];

[1980] argc-=index;

[1981] // if service connectivity is specified, then we assume we

[1982] // are a service provider. We allow two portions in the

[1983] // connectivity specification. the first is the host, and the

[1984] // second is the service. This allows us to use things like:

[1985] // 127.0.0.1:999

[1986] // /local:/tmp/unix_domain_socket

[1987] // /fifo:/tmp/named_pipe

[1988] // /mmap:/path/filename

[1989] // /phone:609-924-07305

[1990] // /email:cjn@gtlinc.com

[1991] // /pager:609-924-7305

[1992] // /fax:609-924-7306

[1993] if(service_connectivity) {
a. service=computer=service_connectivity;
b. while(*service && *service != ':') ++service;
c. if(*service) { *service='\0'; ++service; }

[1994] }

[1995] // when the command line contains an INET specification, for the

[1996] // primitive, we automatically register the service with tds.

[1997] // this is an implementation issue.

[1998] if(computer && service && service_name && strcmp(service_primitive,"INET")==0)

[1999] {
a. libthandle = dlopen("libtcs.so.1.0",RTLD_LAZY);
b. if(!libthandle) {
c. fprintf(stderr,"configuration error on TCS\n");
d. fprintf(stderr,"error condition: %s\n",dlerror());
e. exit(1);

```
f.  }
g.  tcp_accept=dlsym(libthandle,"tcp_accept2");
h.  if(!tcp_accept) {
i.  fprintf(stderr,"TCS configuration error for supporting routines.\n");
j.  exit(1);
k.  }
l.  if( strcasecmp(service_name,"tds")==0) {
m.  void *handle;
n.  int (*tds)(int, char *);
o.  FILE *fp=fopen("/tmp/tds","a");
p.  fprintf(fp,"inet %s %s\n", computer, service);
q.  fclose(fp);
r.  handle=dlopen("../tdsd/libtds.so.1.0",RTLD_LAZY);
s.  if( handle ) {
        i.  char buffer[2048];
        ii. sfsprintf(buffer,2048,"command=query name=tds\n");
        iii. tds=dlsym(handle,"tds");
        iv. if( tds ) (*tds)(0,buffer);
        v.  dlclose(handle);
t.  }
u.  else fprintf(stderr,"cannot find libtds.so.1.0\n");
v.  }
w.  else
x.  {
y.  char gbuff[2048];
z.  int  (*tds)(int,char*);
aa. void *handle;
bb. handle=dlopen("../tdsd/libtds.so.1.0",RTLD_LAZY);
cc. sfsprintf(gbuff,2048,"command=register name=\"%s\" location=\"%s\" physical=\"%s:%s\" primitive=\"%s\"\nQUIT\n",
    service_name, service_location, computer, service, service_primitive);

[2000]  log_f.message(gbuff,0);
a.  if( handle ) {
        i.  tds=dlsym(handle,"tds");
        ii. if(!tds)
        iii. {
        iv. fprintf(stderr,"missing tds. error [%s]\n", dlerror());
        v.  exit(1);
        vi. }
[2001]  log_f.message("attempting to register service.",0);
[2002]  log_f.message(gbuff,0);
        i.  (*tds)(0,gbuff);
        ii. close(handle);
b.  }
[2003]  else fprintf(stderr,"cannot register service. missing libtds\n");
a.  }
b.  (*tcp_accept)(argc, computer, service, service_location, service_name);
[2004]  }
```

[2005] while(1) {

a. handle = dlopen(service_location == NULL ? argv[1] : service_location, RTLD_LAZY);

b. if(! handle) break;

c. if(! service_name) service_name=argv[2];

d. function = dlsym(handle, service_name);

e. if(! function) { dlclose(handle); break; }

f. return ((*function) (argc, argv));

[2006] }

[2007] if(service_name && ! service_location) {

a. FILE *fp=fopen("/tmp/tds","r");

b. while(fscanf(fp,"%s",buffer) == 1) {

c. if(strcmp(buffer,"inet")==0) {

i. memset(buffer,'\0',2048);

ii. fscanf(fp,"%s",buffer);

iii. computer=malloc(strlen(buffer)+1);

iv. strcpy(computer,buffer);

v. memset(buffer,'\0',2048);

vi. fscanf(fp,"%s",buffer);

vii. service=malloc(strlen(buffer)+1);

viii. strcpy(service,buffer);

ix. break;

d. }

e. }

f. if(computer && service) {

g. int fds1[2];

h. int fds2[2];

i. int n;

j. char *bp;

k. char gbuff[2048];

l. pipe(&fds1);

m. pipe(&fds2);

n. sprintf(gbuff,"command=query name=%s\n", service_name);

[2008] log_f.message("sending request",0);

[2009] log_f.message(gbuff,0);

a. if(write(fds1[1],gbuff,strlen(gbuff)) != strlen(gbuff))

i. fprintf(stderr,"write to pipe failed errno=%d\n", errno);

b. close(fds1[1]);

c. client_gl(computer, service, fds1[0], fds2[1]);

d. log_f.message("client_gl completed",0);

e. set_nonblocking(fds2[0]);

f. memset(buffer,'\0',2048);

g. read(fds2[0],buffer,2048);

h. log_f.message("response is",0);

i. log_f.message(buffer,0);

j. bp=buffer;

k. while(bp) {

i. name=NULL;

ii. value=NULL;

```
            iii.  bp=getnvpair(bp, &name, &value);
            iv.   if( ! name || ! value) break;
            v.    log_f.message("processing name",0);
            vi.   log_f.message(name,0);
            vii.  if(strcmp(name,"location")==0) {
            viii. service_location=malloc(strlen(value)+1);
            ix.   strcpy(service_location,value);
            x.    }
            xi.   else if(strcmp(name,"primitive")==0) {
            xii.  service_primitive=malloc(strlen(value)+1);
            xiii. strcpy(service_primitive,value);
            xiv.  }
            xv.   else if(strcmp(name,"physical")==0) {
            xvi.  service_connectivity=malloc(strlen(value)+1);
            xvii. strcpy(service_connectivity,value);
            xviii. }
l.      }
m.      close(fds1[0]);
n.      close(fds1[1]);
o.      close(fds2[0]);
p.      close(fds2[1]);
q.      if(service_connectivity) {
            i.    service=computer=service_connectivity;
            ii.   while(*service && *service != ':') ++service;
            iii.  if( *service ) { *service='\0'; ++service; }
r.      }
s.      else log_f.message("missing service connectivity",-1);
t.      client_gl(computer, service, 0, 1);
u.      }
[2010]  }
[2011]  return(0);
[2012]  }
[2013]  int get_tds_connection_info(char **computer, char **service)
[2014]  {
[2015]  char buffer[2048];
[2016]  FILE *fp=fopen("/tmp/tds","r");
[2017]  while( fscanf(fp,"%s",buffer) == 1 ) {
a.      if(strcmp(buffer,"inet")==0) {
b.      memset(buffer,'\0',2048);
c.      fscanf(fp,"%s",buffer);
d.      *computer=malloc(strlen(buffer)+1);
e.      strcpy(*computer,buffer);
f.      memset(buffer,'\0',2048);
g.      fscanf(fp,"%s",buffer);
h.      *service=malloc(strlen(buffer)+1);
i.      strcpy(*service,buffer);
j.      break;
```

k. }

[2018] }

[2019] fclose(fp);

[2020] }

[2021] Program Listing 9.5 generic front end loader service sendreceive.c

```
[2022] #include <stdio.h>
[2023] #include <limits.h>
[2024] #include <errno.h>
[2025] #include "log.h"
[2026] int sendreceive(char *computer, char *service, char *request, int size, char *response, int *rsize)
[2027] {
[2028] int fds1[2];
[2029] int fds2[2];
[2030] int n;
[2031] char *bp;
[2032] char gbuff[PIPE_BUF];
[2033] log_f.enter("sendreceive",0);
[2034] if(.(!computer) || (!service) || (!request) || (!size) || (!response) || (!rsize)) {
a. log_f.message("bad parameters",-1);
b. log_f.exit("sendreceive",-1);
c. return(-1);
[2035] }
[2036] if(size > PIPE_BUF) {
a. log_f.message("send message > PIPE_BUF size",-1);
b. log_f.exit("sendreceive",-1);
c. return(-1);
[2037] }
[2038] if( pipe( &fds1 ) ) {
a. log_f.message("out of resources",-1);
b. log_f.exit("sendreceive",-1);
c. return(-1);
[2039] }
[2040] if(pipe( &fds2 )) {
a. close(fds1[0]);
b. close(fds1[1]);
c. log_f.message("out of resources",-1);
d. log_f.exit("sendreceive",-1);
e. return(-1);
```

[2041] }

[2042] if(write(fds1[1],request,size) != size) {

a. fprintf(stderr,"write to pipe failed errno=%d\n", errno);

b. close(fds1[0]);

c. close(fds1[1]);

d. close(fds2[0]);

e. close(fds2[1]);

f. log_f.message("write to pipe failed",errno);

g. log_f.exit("sendreceive",-1);

h. return(-1);

[2043] }

[2044] close(fds1[1]);

[2045] client_gl2(computer, service, fds1[0], fds2[1]);

[2046] n=peek(fds2[0]);

[2047] if(n > *rsize) n=*rsize;

[2048] memset(response,'\0',*rsize);

[2049] *rsize=0;

[2050] if(n > 0 && peek(fds2[0]) > 0) *rsize=read(fds2[0],response,n);

[2051] close(fds1[0]);

[2052] close(fds2[0]);

[2053] close(fds2[1]);

[2054] log_f.message("normal return",0);

[2055] log_f.exit("sendreceive",*rsize);

[2056] return(*rsize);

[2057] }

[2058] ## Program Listing 10.1 thread connection service - talk2.c

[2059] #include <stdio.h>

[2060] #include <stdlib.h>

[2061] #include <sys/select.h>

[2062] #include <sys/time.h>

[2063] #include <dlfcn.h>

[2064] #include <errno.h>

[2065] int talk2(int *fd1_set, int *fd2_set)

[2066] {

[2067] fd_set read_fs, error_fs;

[2068] int nfds, r;

```
[2069]  int   adminfd=-1;
[2070]  struct  timeval t;
[2071]  int   (*function)(void) = NULL;
[2072]  int   timeout=10;
[2073]  memset( &t, \0', sizeof(struct timeval));
[2074]  if((function = dlsym(NULL,"get_tcs_timeout")) != NULL) timeout = (*function)();
[2075]  t.tv_sec = timeout;
[2076]  if( fd2_set == NULL ) while( participant(fd1_set[0],fd1_set[1])>0) ;
[2077]  else while( 1 ) {
a.      t.tv_sec = timeout;
b.      FD_ZERO( &read_fs );
c.      FD_ZERO( &error_fs );
d.      nfds=0;
e.      if(fd1_set) {
f.      FD_SET( fd1_set[0], &read_fs );
g.      FD_SET( fd1_set[0], &error_fs );
h.      nfds=fd1_set[0];
i.      }
j.      if(fd2_set) {
k.      FD_SET( fd2_set[0], &read_fs );
l.      FD_SET( fd2_set[0], &error_fs );
m.      if( fd2_set[0] > nfds ) nfds=fd2_set[0];
n.      }
o.      if(!nfds) break;
p.      ++nfds;
q.      if( (r=select(nfds,&read_fs,NULL,&error_fs,&t)) < 0 ) return( -1 );
r.      if( r == 0 ) break;
s.      if(fd1_set) {
t.      if( FD_ISSET(fd1_set[0], &read_fs)) {
                i.  int t=participant(fd1_set[0],fd1_set[1]);
                ii. if(t <= 0 ) { shutdown(fd1_set[1],1); fd1_set=NULL; }
u.      }
v.      else if( FD_ISSET(fd1_set[0], &error_fs)) break;
w.      }
x.      if(fd2_set) {
y.      if( FD_ISSET(fd2_set[0], &read_fs)) {
                i.  int t;
                ii. t=participant(fd2_set[0],fd2_set[1]);
                iii. if( t <= 0 ) { shutdown(fd2_set[1],0); fd2_set=NULL; }
z.      }
aa.     else if( FD_ISSET(fd2_set[0], &error_fs)) break;
bb.     }
[2078]  }
[2079]  return(0);
[2080]  }
```

[2081] Program Listing 10.2 thread connection service - participant.c

```
[2082]  #include <stdio.h>
[2083]  #include <errno.h>
[2084]  #include <fcntl.h>
[2085]  int participant(int fd1, int fd2)
[2086]  {
[2087]  static char *buffer;
[2088]  static int   buffer_size;
[2089]  int n=peek(fd1);
[2090]  int n1;
[2091]  if(! buffer_size) { buffer_size=4098; buffer=(char *)malloc(buffer_size); }
[2092]  if( n > 0 ) {
        a.  if( n > buffer_size ) {
        b.  if(buffer) free(buffer);
        c.  buffer_size=n+1;
        d.  buffer=(char *)malloc(buffer_size);
        e.  }
        f.  memset(buffer,'\0',buffer_size);
        g.  n1=n;
        h.  n=readn(fd1,buffer,n);
        i.  if( n == -1 ) return(0);
        j.  if(n == 0 )return(0);
[2093]  writenagain:
        a.  errno=0;
        b.  n1=writen(fd2,buffer,n);
        c.  if(n1 != n ) {
        d.  if(n1 == -1 && errno == EAGAIN) {
                i.   int nfds;
                ii.  int r;
                iii. fd_set write_set, error_set;
                iv.  nfds=fd2+1;
                v.   FD_ZERO( &write_set);
                vi.  FD_ZERO( &error_set);
                vii. FD_SET( fd2, &write_set);
                viii. FD_SET( fd2, &error_set);
                ix.  r=select(nfds,NULL,&write_set,&error_set,NULL);        if(r == 1 ) {
                x.   if(FD_ISSET(fd2,&write_set)) goto writenagain;
                xi.  }
                xii. else if(r == -1 && errno == EINTR) goto writenagain;
        e.  }
        f.  }
[2094]  }
```

[2095] return(n);

[2096] }

[2097] Program Listing 10.3 thread connection service - tcp_accept2.c

```
[2098] #include <stdio.h>
[2099] #include <unistd.h>
[2100] #include <stdlib.h>
[2101] #include <sys/socket.h>
[2102] #include <sys/types.h>
[2103] #include <sys/time.h>
[2104] #include <fcntl.h>
[2105] #include <errno.h>
[2106] #include <netdb.h>
[2107] #include <dlfcn.h>
[2108] int
[2109] tcp_accept2(int argc, char *computer, char *service, char *service_location, char *service_name)
[2110] {
[2111] int         listenfd;
[2112] int         adminlistenfd=-1;
[2113] int         connfd;
[2114] socklen_t   addrlen=0, len;
[2115] struct sockaddr  *cliaddr;
[2116] time_t            ticks;
[2117] char        buff[2048];
[2118] void        *libhandle;
[2119] int         (*function)(int,void *);
[2120] int         (*timeout_function)(void) = NULL;
[2121] int         (*adminfunction)(int) =NULL;
[2122] int         *timeout_value;
[2123] int         rc=0;
[2124] if( (libhandle = dlopen(service_location,RTLD_LAZY|RTLD_GLOBAL)) == NULL) {
a.  char *cp=dlerror();
b.  fprintf(stderr,"dlopen %s failed errno=%d\n", service_location,errno);
c.  fprintf(stderr,"error message [%s]\n", cp);
[2125] }
[2126] if( (function  = dlsym(libhandle, service_name)) == NULL) {
```

```
a.  fprintf(stderr,"dlsym of function [%s] failed errno=%d\n", service, errno);
b.  fprintf(stderr,"service not available\n");
c.  exit(0);
[2127]  }
[2128]  sprintf(buff,"%s_timeout",service_name);
[2129]  timeout_function = dlsym(libhandle,buff);
[2130]  sprintf(buff,"%s_timeout_value",service_name);
[2131]  timeout_value = dlsym(libhandle,buff);
[2132]  sprintf(buff,"%s_admin",service_name);
[2133]  adminfunction=dlsym(libhandle,buff);
[2134]  if( strcmp(computer,"")==0) listenfd = tcp_listen(NULL, service, &addrlen);
[2135]  else listenfd = tcp_listen(computer, service, &addrlen);
[2136]  len=addrlen;
[2137]  cliaddr = malloc(addrlen);
[2138]  for( ; 1 ; ) {
a.  struct  timeval t;
b.  int    r, nfds;
c.  fd_set      read_set, write_set, error_set;
d.  memset( &t, '\0', sizeof(struct timeval));
e.  memset( cliaddr, '\0', addrlen);
f.  t.tv_sec = timeout_value ? *timeout_value : 30;
g.  FD_ZERO( &read_set );
h.  FD_ZERO( &write_set );
i.  FD_ZERO( &error_set );
j.  FD_SET( listenfd,  &read_set );
k.  FD_SET( listenfd,  &error_set );
l.  nfds=listenfd;
m.  if(adminfunction && adminlistenfd == -1)
n.  adminlistenfd=(*adminfunction)(0);
o.  if(adminlistenfd>0) {
p.  FD_SET( adminlistenfd,  &read_set );
q.  FD_SET( adminlistenfd,  &error_set );
r.  nfds=nfds>adminlistenfd ? nfds : adminlistenfd;
s.  }
t.  ++nfds;
u.  r=select(nfds,&read_set,NULL,&error_set,&t);
v.  if( r < 0 ) {
w.  if(errno == EINTR ) continue;
x.  fprintf(stderr,"tcp_accept2: select failed errno: %d\n",errno);
y.  exit(1);
z.  }
aa.  if(r == 0 ) { if(timeout_function) (*timeout_function)(); continue; }
bb.  if( adminfunction && adminlistenfd > 0) {
cc.  if( FD_ISSET(adminlistenfd, &read_set)) {
        i.  connfd = accept(adminlistenfd, cliaddr, &len );
```

ii. if(connfd == -1) { if(errno == EINTR) continue; break; }
iii. rc=(*adminfunction)(connfd);
iv. shutdown(connfd,2);
v. close(connfd);
vi. if(rc) break;

dd. }
ee. }
ff. if(FD_ISSET(listenfd, &read_set)) {
gg. connfd = accept(listenfd, cliaddr, &len);
hh. if(connfd == -1) { if(errno == EINTR) continue; break; }
ii. if(function)

[2139] { (*function)(connfd,cliaddr); shutdown(connfd,2); close(connfd); }

a. else {
i. ticks=time(NULL);
ii. sprintf(buff,"%.24s\n",ctime(&ticks));
iii. write(connfd,buff,strlen(buff));
iv. shutdown(connfd,2);
v. close(connfd);

b. }
c. }

[2140] }

[2141] return(rc);

[2142] }

[2143] Program Listing 10.4 thread connection service - tcp_connect.c

[2144] #include "services.h"

[2145] int tcp_connect(const char *host, const char *serv)

[2146] {

[2147] int sockfd, n;

[2148] struct addrinfo hints, *res, *ressave;

[2149] bzero(&hints,sizeof(struct addrinfo));

[2150] hints.ai_family=AF_UNSPEC;

[2151] hints.ai_socktype=SOCK_STREAM;

[2152] if((n=getaddrinfo_gamix(host, serv, &hints, &res)) != 0) {

a. fprintf(stderr,"tcp_connect: getaddrinfo failed errno=%d\n", errno);
b. exit(1);

[2153] }

[2154] ressave=res;

[2155] do {

a. sockfd=socket(res->ai_family, res->ai_socktype, res->ai_protocol);
b. if(sockfd < 0) continue;

```
        c.  if( connect(sockfd,res->ai_addr,res->ai_addrlen) == 0) break;
        d.  close(sockfd);
[2156]      } while( (res=res->ai_next) != NULL);
[2157]      if(res == NULL ) {
        a.  fprintf(stderr,"tcp_connect error for %s, %s\n", host,serv);
        b.  return(-1);
[2158]      }
[2159]      freeaddrinfo(ressave);
[2160]      return(sockfd);
[2161]      }
```

[2162] Program Listing 10.5 thread connection service - tcp_listen.c

```
[2163]  #include <stdio.h>
[2164]  #include <unistd.h>
[2165]  #include <stdlib.h>
[2166]  #include <sys/types.h>
[2167]  #include <sys/time.h>
[2168]  #include <fcntl.h>
[2169]  #include <errno.h>
[2170]  #include <netdb.h>
[2171]  #include <dlfcn.h>
[2172]  #if ! defined(LISTENQ)
[2173]  #define LISTENQ     256
[2174]  #endif
[2175]  int
[2176]  tcp_listen(char *host, char *serv, socklen_t *addrlenp)
[2177]  {
[2178]  int listenfd, n;
[2179]  const int on=1;
[2180]  struct addrinfo hints, *res, *ressave;
[2181]  bzero( &hints, sizeof(struct addrinfo));
[2182]  hints.ai_flags = AI_PASSIVE;
[2183]  hints.ai_family= AF_UNSPEC;
[2184]  hints.ai_socktype=SOCK_STREAM;
[2185]  if((n=getaddrinfo_gamix(host, serv, &hints, &res)) != 0 )
[2186]  { fprintf(stderr,"tcp_listen: getaddrinfo failed errno=%d\n", errno); exit(1); }
```

```
[2187]      ressave=res;
[2188]      do
[2189]      {
        a.  listenfd = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
        b.  if( listenfd < 0 ) continue;
        c.  setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
        d.  if( bind(listenfd, res->ai_addr, res->ai_addrlen) == 0 ) break;
        e.  close(listenfd);
[2190]      } while((res=res->ai_next) != NULL );
[2191]      if(res == NULL ) {
        a.  fprintf(stderr,"tcp_listen: error for %s, %s\n", host,serv);
        b.  exit(1);
[2192]      }
[2193]      listen(listenfd,LISTENQ);
[2194]      if(addrlenp) *addrlenp=res->ai_addrlen;
[2195]      freeaddrinfo(ressave);
[2196]      return(listenfd);
[2197]      }
```

[2198] Program Listing 11.1 supporting functions - reaper.c

```
[2199]  #include <signal.h>
[2200]  #include <sys/types.h>
[2201]  #include <sys/wait.h>
[2202]  void reaper(int signo)
[2203]  {
[2204]  pid_t pid; int stat;
[2205]  while((pid=waitpid(-1,&stat,WNOHANG))>0)     ;
[2206]  return;
[2207]  }
```

[2208] Program Listing 12.1 supporting service - cat_service.c

```
[2209]  #include <stdio.h>
[2210]  #include <fcnlt.h>
[2211]  #include <errno.h>
[2212]  int
[2213]  cat_service(int argc, char **argv)
[2214]  {
[2215]  int n, fd[2], fdi=0, fdo=1, didopen=0;
```

```
[2216]      char buffer[2048];
[2217]      buffer[0]='\0';
[2218]      if(argv == NULL ) { fdo=argc;
        a.  n=readline(fdo,buffer,2048);
        b.  if(n>0) buffer[n-1]='\0';
[2219]      } else strcpy(buffer, argv[1]);
[2220]      if((fdi=open(buffer,O_RDONLY)) >=0 ) { fd[0]=fdi; fd[1]=fdo; ++didopen; }
[2221]      talk2(fd,NULL);
[2222]      if(didopen) close(fdi);
[2223]      return(n);
[2224]      }
```

[2225] Program Listing 12.2 supporting service - echo_service.c

```
[2226]  #include <stdio.h>
[2227]  #include <fcnlt.h>
[2228]  #include <errno.h>
[2229]  int echo_service(int fd)
[2230]  {
[2231]  ssize_t n;
[2232]  char buffer[2048];
[2233]  for(;1;) if( (n=readline(fd,buffer,2048))<=0) return; else writen(fd,buffer,n);
[2234]  return(0);
[2235]  }
```

[2236] Program Listing 12.3 supporting service - daytime_service.c

```
[2237]  #include <stdio.h>
[2238]  int
[2239]  daytime_service(int argc, char **argv)
[2240]  {
[2241]  time_t      ticks = time(NULL);
[2242]  int    n;
[2243]  int    fd_out=0;
[2244]  char  buffer[512];
[2245]  if( argv == NULL ) fd_out=argc;
[2246]  sprintf(buffer,"%.24s\n", ctime(&ticks));
```

```
[2247]  n = write(fd_out,buffer,strlen(buffer));
[2248]  return( n );
[2249]  }
```

[2250] Program Listing 12.4 supporting service - ksh_service.c

```
[2251]  #include <stdio.h>
[2252]  #include <sys/ioctl.h>
[2253]  #include <unistd.h>
[2254]  #include <dlfcn.h>
[2255]  #include <sys/signal.h>
[2256]  int
[2257]  ksh_service(int fd, char *argv[])
[2258]  {
[2259]  void *handle= dlopen("./libservices.so.1.0", RTLD_LAZY);
[2260]  int    (*sigc)(int,void *);
[2261]  int fd_in=0;
[2262]  int fd_out=1;
[2263]  int fd_err=2;
[2264]  char buffer[2048];
[2265]  char criteria[2048];
[2266]  char *n, *v;
[2267]  int  ni;
[2268]  int  didfork=0;
[2269]  char *cp;
[2270]  char *physical=NULL;
[2271]  char *primitive=NULL;
[2272]  char *name=NULL;
[2273]  char *location=NULL;
[2274]  char *argv2[3];
[2275]  pid_t pid;
[2276]  if( argv == NULL ) {
        a.  fd_in=fd;
        b.  fd_out=fd;
        c.  fd_err=fd;
        d.  if( handle ) {
        e.  if( (sigc=dlsym(handle,"sig_child")) ) {
                i.  signal( SIGCHLD, sigc);
```

```
                ii.   pid=fork();
                iii.  if( pid != 0 ) { dlclose(handle); return(0); }
                iv.   ++didfork;
        f.    }
        g.    }
        h.    else fprintf(stderr,"do not have a handle\n");
[2277]        }
[2278]        else fprintf(stderr,"argv is NOT null\n");
[2279]        #if defined(TIOCSTTY)
[2280]        if( ioctl(fd,TIOCSTTY,(char *NULL)) == -1 ) return(-1);
[2281]        #endif
[2282]        daemon(1,0);
[2283]        while(1)    {
[2284]        dup2(fd_in,0);
        a.    dup2(fd_out,1);
        b.    dup2(fd_err,2);
        c.    // given some criteria of what we want the mail
        d.    // service to do, see if there is a registered service.
        e.    // typically, this would mean we get something like:
        f.    // name="Charles Northrup" company="Global Technologies"
        g.    // and tds would fine the email-id.
        h.    memset(criteria,'\0',2048);
        i.    ni=readline(0,criteria,2048);
        j.    if(ni > 0 ) criteria[ni-1]='\0';
        k.    else break;
        l.    memset(buffer,'\0',2048);
        m.    tds_query_p(criteria,buffer,2048);
        n.    tds_get_nlpp(buffer, &name, &location, &primitive, &physical, NULL, NULL);
        o.    execlp("ksh","/bin/ksh",NULL,NULL);
        p.    break;
[2285]        }
[2286]        if( didfork ) exit(0);
[2287]        return(0);
[2288]        }
```

[2289] Program Listing 12.5 mail service - mail_service.c

```
[2290]  #include <stdio.h>
[2291]  #include <sys/ioctl.h>
[2292]  #include <unistd.h>
[2293]  #include <dlfcn.h>
[2294]  #include <sys/signal.h>
```

```
[2295]  int mail_service(int fd, char *argv[])
[2296]  {
[2297]  void *handle= dlopen("./libservices.so.1.0", RTLD_LAZY);
[2298]  int    (*sigc)(int,void *);
[2299]  int fd_in=0;
[2300]  int fd_out=1;
[2301]  int fd_err=2;
[2302]  char buffer[2048];
[2303]  char criteria[2048];
[2304]  char *n, *v;
[2305]  int  ni;
[2306]  int  didfork=0;
[2307]  char *cp;
[2308]  char *physical=NULL;
[2309]  char *primitive=NULL;
[2310]  char *name=NULL;
[2311]  char *location=NULL;
[2312]  char *argv2[3];
[2313]  pid_t pid;
[2314]  if( argv == NULL ) {
[2315]  fd_in=fd;
        a.  fd_out=fd;
        b.  fd_err=fd;
        c.  if( handle ) {
        d.  if( (sigc=dlsym(handle,"sig_child")) ) {
                i.   signal( SIGCHLD, sigc);
                ii.  pid=fork();
                iii. // if we are the parent, then
                iv.  // close the handle to the lib and
                v.   // return.
                vi.  if( pid != 0 )
                vii. { dlclose(handle); return(0); }
                viii. ++didfork;
        e.  }
        f.  }
        g.  else fprintf(stderr,"do not have a handle\n");
[2316]  }
[2317]  else fprintf(stderr,"argv is NOT null\n");
[2318]  #if defined(TIOCSTTY)
[2319]  if( ioctl(fd,TIOCSTTY,(char *NULL)) == -1 ) return(-1);
```

```
[2320]  #endif
[2321]  daemon(1,0);
[2322]  while(1)
[2323]  {   dup2(fd_in,0);
a.  dup2(fd_out,1);
b.  dup2(fd_err,2);
c.  // given some criteria of what we want the mail
d.  // service to do, see if there is a registered service.
e.  // typically, this would mean we get something like:
f.  // name="Charles Northrup" company="Global Technologies"
g.  // and tds would fine the email-id.
h.  memset(criteria,'\0',2048);
i.  ni=readline(0,criteria,2048);
j.  if(ni > 0 ) { criteria[ni-1]='\0'; ni--; }
k.  if(ni == 0 ) break;
l.  memset(buffer,'\0',2048);
m.  tds_query_p(criteria,buffer,2048);
n.  tds_get_nlpp(buffer, &name, &location, &primitive, &physical, NULL, NULL);
o.  while( primitive && physical ) {
p.  if( strcmp(primitive,"email") != 0 ) break;
q.  execlp("mail","mail",physical,NULL);
r.  break;
s.  }
t.  break;
[2324]  }
[2325]  if( didfork ) exit(0);
[2326]  return(0);
[2327]  }
```

[2328] Program Listing 13.1 TDS supporting functions - tds_query_p.c

```
[2329]  #include <stdio.h>
[2330]  #include <errno.h>
[2331]  #include <malloc.h>
[2332]  extern char *getnvpair(char *, char **, char **);
[2333]  int  tds_get_nvpair(char *answer, char *name, char **value)
[2334]  {
[2335]  int length=strlen(answer)+1;
[2336]  char *alt, *cp;
[2337]  char *n, *v;
[2338]  alt=malloc(length);
[2339]  memset(alt,'\0',length);
```

[2340] memcpy(alt,answer,length-1);

[2341] cp=alt;

[2342] while(*cp)

[2343] {

a. n=v=NULL;

b. cp=getnvpair(cp, &n, &v);

c. if(!n || !v) break;

d. if(strcmp(n,name)==0) {

[2344] if(value) {

[2345] *value=malloc(strlen(v)+1);

i. memset(*value,'\0',strlen(v)+1);

ii. memcpy(*value,v,strlen(v));

iii. break;

b. }

c. }

[2346] }

[2347] free(alt);

[2348] }

[2349] int tds_get_nlpp(char *answer,char **name, char **location, char **primitive, char **physical, int *argc, char **args)

[2350] {

[2351] char *cp;

[2352] char *n, *v;

[2353] char *g=malloc(strlen(answer)+1);

[2354] strcpy(g,answer);

[2355] cp=g;

[2356] while(*cp)

[2357] {

a. n=v=NULL;

b. cp=getnvpair(cp, &n, &v);

c. if(!n || !v) break;

d. if(name && strcmp(n,"name")==0)

[2358] { *name=malloc(strlen(v)+1); strcpy(*name, v); }

a. else if(location && strcmp(n,"location")==0)

[2359] { *location=malloc(strlen(v)+1); strcpy(*location, v); }

a. else if(primitive && strcmp(n,"primitive")==0)

[2360] { *primitive=malloc(strlen(v)+1); strcpy(*primitive, v); }

a. else if(physical && strcmp(n,"physical")==0)

[2361] { *physical=malloc(strlen(v)+1); strcpy(*physical, v); }

a. else if(args && strcmp(n,"arg")==0)

b. {

c. char *p=malloc(strlen(v)+1);

```
d.    memset(p,'\0',strlen(v)+1);
e.    memcpy(p,v,strlen(v));
f.    if( ! args )
g.    { args=malloc(sizeof(char *)*10); memset(args,'\0',sizeof(char *)*10);}
h.    if( ! argc )
i.    { argc=malloc(sizeof(int)); *argc=0; }
j.    args[*argc]=p;
k.    ++(*argc);
l.    }
[2362]    }
[2363]    if(g) free(g);
[2364]    return(0);
[2365]    }
[2366]    int tds_query_p(char *criteria, char *answer, int answer_size)
[2367]    {
[2368]    FILE *fp=fopen("/tmp/tds","r");
[2369]    char *computer=NULL, *service=NULL;
[2370]    char buffer[2048];
[2371]    // we first assume that if a tds is running, it has the
[2372]    // identifier info in the /tmp/tds file.  this could
[2373]    // be moved to /var/adm/tds for consistency across implementations,
[2374]    // but, that is an implementation detail.
[2375]    memset(buffer,'\0',2048);
[2376]    while( fscanf(fp,"%s",buffer) == 1 ) {
a.    if(strcmp(buffer,"inet")==0) {
b.    memset(buffer,'\0',2048);
c.    fscanf(fp,"%s",buffer);
d.    computer=malloc(strlen(buffer)+1);
e.    strcpy(computer,buffer);
f.    memset(buffer,'\0',2048);
g.    fscanf(fp,"%s",buffer);
h.    service=malloc(strlen(buffer)+1);
i.    strcpy(service,buffer);
j.    break;
k.    }
[2377]    }
[2378]    fclose(fp);
[2379]    // assuming we found a computer and a service, then
[2380]    // we shall send it a message.
[2381]    if( computer && service ) {
a.    int fds1[2];
b.    int fds2[2];
```

```
c.  int n;
d.  char *bp;
e.  pipe( &fds1 );
f.  pipe( &fds2 );
g.  sprintf(buffer,"QUERY %s\nQUIT\n", criteria);
h.  if( write(fds1[1],buffer,strlen(buffer)) != strlen(buffer)) {
i.  fprintf(stderr,"write to pipe failed errno=%d\n", errno);
j.  return(-1);
k.  }
l.  close(fds1[1]);
m.  client_gl(computer, service, fds1[0], fds2[1]);
n.  set_nonblocking(fds2[0]);
o.  memset(answer,'\0',answer_size);
p.  n=readline(fds2[0],answer,2048);
q.  if( n > 0 ) if( answer[n-1] == '\n' ) answer[n-1]='\0';
r.  close(fds1[0]);
s.  close(fds2[0]);
t.  return(strlen(answer));
[2382]      }
[2383]      return(-1);
[2384]      }
[2385]      int tds_query_c(char *criteria, char **answer, int *answer_size)
[2386]      {
[2387]      char *computer=NULL, *service=NULL;
[2388]      char *buffer=NULL;
[2389]      int fds1[2];
[2390]      int fds2[2];
[2391]      int n;
[2392]      char *bp;
[2393]      get_tds_connection_info(&computer, &service);
[2394]      if( ! computer || ! service ) return(-1);
[2395]      buffer=malloc(strlen(criteria)+200);
[2396]      memset(buffer,'\0',strlen(criteria)+200);
[2397]      sprintf(buffer,"QUERY %s\nQUIT\n", criteria);
[2398]      pipe( &fds1 );
[2399]      pipe( &fds2 );
[2400]      if( write(fds1[1],buffer,strlen(buffer)) != strlen(buffer))
[2401]      { fprintf(stderr,"write to pipe failed errno=%d\n", errno); return(-1); }
[2402]      client_gl(computer, service, fds1[0], fds2[1]);
[2403]      set_nonblocking(fds2[0]);
[2404]      memset(*answer,'\0',*answer_size);
```

```
[2405]  waitread(fds2[0]);
[2406]  n=peek(fds2[0]);
[2407]  *answer=malloc(n+1);
[2408]  memset(*answer,'\0',n+1);
[2409]  n=readn(fds2[0],*answer,n);
[2410]  if( n > 0 ) if( (*answer)[n-1] == '\n' ) (*answer)[n-1]='\0';
[2411]  close(fds1[0]);
[2412]  close(fds1[1]);
[2413]  close(fds2[0]);
[2414]  close(fds2[1]);
[2415]  if(answer_size)  *answer_size=n;
[2416]  free(computer);
[2417]  free(service);
[2418]  free(buffer);
[2419]  return(0);
[2420]  }
```

[2421] Program Listing 13.2 TDS supporting functions - tds_register_p.c

```
[2422]  #include <stdio.h>
[2423]  #include <errno.h>
[2424]  #include <malloc.h>
[2425]  typedef struct _tdsmsg {
[2426]  int    msize;
[2427]  int    bsize;
[2428]  char  *buffer;
[2429]  } tdsmsg_t;
[2430]  static tdsmsg_t  tdsmsg;
[2431]  static int tds_strlen(char *m)
[2432]  {
[2433]  int s=0;
[2434]  char *cp=m;
[2435]  if( !m )     return(0);
[2436]  for(cp=m;*cp;++cp) ++s;
[2437]  return(s);
```

```
[2438]      }
[2439]      int tds_seterror(char *e)
[2440]      {
[2441]      int    msize=0;
[2442]      char  *cp;
[2443]      if( ! e )      return(-1);
[2444]      msize=tds_strlen(e);
[2445]      if(!msize)   return(0);
[2446]      if( tdsmsg.bsize < msize ) {
        a.  if( tdsmsg.buffer ) free(tdsmsg.buffer);
        b.  tdsmsg.buffer = (char *)malloc(msize+1);
        c.  tdsmsg.bsize=msize+1;
[2447]      }
[2448]      if( ! tdsmsg.buffer) return(-1);
[2449]      memset(tdsmsg.buffer,'\0',tdsmsg.bsize);
[2450]      memcpy(tdsmsg.buffer,e,msize);
[2451]      tdsmsg.msize=msize;
[2452]      return(0);
[2453]      }
[2454]      int
[2455]      tds_geterror(char *m, int *size)
[2456]      {
[2457]      if( !m || ! size) return(-1);
[2458]      if( ! tdsmsg.buffer ) { *size=0; *m='\0'; return(0); }
[2459]      if( *size < tdsmsg.msize ) { *size=tdsmsg.msize+1; return(-1); }
[2460]      memcpy(m, tdsmsg.buffer, tdsmsg.msize);
[2461]      *size=tdsmsg.msize;
[2462]      m[*size]='\0';
[2463]      return(0);
[2464]      }
[2465]      int
[2466]      tds_register(char *registration, char *results, int *size)
[2467]      {
[2468]      char *computer=NULL, *service=NULL;
[2469]      char buffer[4096];
[2470]      int pipe1[2];
```

```
[2471]      int pipe2[2];
[2472]      int n;
[2473]      int rsize;
[2474]      char *tds_computer=NULL, *tds_service=NULL;
[2475]      if( get_tds_connection_info(&tds_computer, &tds_service) == -1 ) {
        a.  if( *size ) *size=0; tds_seterror("tds connection failed\n");
        b.  return(-1);
[2476]      }
[2477]      if( ! tds_computer || ! tds_service ) {
        a.  if( *size ) *size=0; tds_seterror("Failed to locate the TDS Server\n");
        b.  return(-1);
[2478]      }
[2479]      if( pipe( &pipe1 ) != 0 ) {
        a.  if( *size ) *size=0;
[2480]      tds_seterror("pipe system call failed. Probably have too many open files.\n");
        a.  return(-1);
[2481]      }
[2482]      if( pipe( &pipe2 ) != 0 ) {
        a.  if( *size ) *size=0;
        b.  tds_seterror("pipe system call failed. Probably have too many open files.\n");
        c.  return(-1);
[2483]      }
[2484]      rsize=tds_strlen(registration)+100;
[2485]      if(rsize >= 4096 ) {
        a.  tds_seterror("Registration request was too long.\nMaximum size is 4096.\n");
        b.  return(-1);
[2486]      }
[2487]      memset(buffer,'\0',rsize+1);
[2488]      sprintf(buffer,"REGISTER %s\nQUIT\n", registration);
[2489]      if( write(pipe1[1],buffer,rsize) != rsize) {
        a.  tds_seterror("Failed to send the registration request to the server.\n");
        b.  if( *size ) *size=0;
        c.  return(-1);
[2490]      }
[2491]      close(pipe1[1]);
[2492]      client_gl(tds_computer, tds_service, pipe1[0], pipe2[1]);
[2493]      memset(buffer,'\0',4096);
[2494]      if( close(pipe2[1]) != 0 )      fprintf(stderr,"failed close of pipe2[1]=%d\n", pipe2[1]);
[2495]      *size = read(pipe2[0],results,*size);
[2496]      close(pipe1[0]);
```

```
[2497]      close(pipe2[0]);
[2498]      return(0);
[2499]      }
[2500]      int tds_register_p(char *registration)
[2501]      {
[2502]      char  results[4096];
[2503]      int    size=4096;
[2504]      int    rc;
[2505]      if( (rc=tds_register(registration, results, &size)) != 0 ) {
        a.  tds_geterror(results, &size);
        b.  fprintf(stderr,"tds_register_p: %s\n", results);
[2506]      }
[2507]      return(rc);
[2508]      }
```

[2509] Program Listing 13.3 TDS supporting functions - getdtscinfo.c

```
[2510]      #include <stdio.h>
[2511]      #include <memory.h>
[2512]      #include <string.h>
[2513]      #if !defined TDS_CONNECTION_PATH
[2514]      #     define TDS_CONNECTION_PATH     "/tmp/tds.cinfo"
[2515]      #endif
[2516]      int get_tds_connection_info(char **computer, char **service)
[2517]      {
[2518]      char buffer[2048];
[2519]      FILE *fp=fopen(TDS_CONNECTION_PATH,"r");
[2520]      if( fp == NULL )return(-1);
[2521]      while( fscanf(fp,"%s",buffer) == 1 ) {
        a.  if(strcmp(buffer,"inet")==0) {
        b.  memset(buffer,'\0',2048);
        c.  fscanf(fp,"%s",buffer);
        d.  *computer=malloc(strlen(buffer)+1);
        e.  strcpy(*computer,buffer);
        f.  memset(buffer,'\0',2048);
        g.  fscanf(fp,"%s",buffer);
        h.  *service=malloc(strlen(buffer)+1);
        i.  strcpy(*service,buffer);
        j.  break;
        k.  }
```

```
[2522]  }
[2523]  fclose(fp);
[2524]  return(0);
[2525]  }
```

[2526] Program Listing 13.4 TDS supporting functions - tds.c

```
[2527]  #include <stdio.h>
[2528]  #include <malloc.h>
[2529]  #include <sys/utsname.h>
[2530]  #include <errno.h>
[2531]  typedef struct _ic_t {
[2532]  char *field_name;
[2533]  char *field_value;
[2534]  int public;
[2535]  struct _ic_t *next;
[2536]  struct _ic_t *last;
[2537]  } ic_t;
[2538]  typedef struct _ic_f {
[2539]  ic_t   *(*constructor)();
[2540]  int     (*destructor)(ic_t *);
[2541]  ic_t   *(*make_ic)(char *, char *);
[2542]  ic_t   *(*locate)(ic_t *, char *, char *);
[2543]  } ic_f_t;
[2544]  typedef struct _ste_t {
[2545]  ic_t   *ic;
[2546]  struct _ste_t      *next;
[2547]  struct _ste_t      *last;
[2548]  } ste_t;
[2549]  typedef struct _ste_base_t {
[2550]  ste_t *first;
[2551]  ste_t *last;
[2552]  } ste_base_t;
[2553]  typedef struct _ste_f_t {
[2554]  int    (*addic) (ste_t *, char *, char *);
[2555]  ste_t  *(*constructor)();
```

```
[2556]      int    (*destructor)(ste_t *);
[2557]      ste_t  *(*locate_by_name_and_location)(char *, char *);
[2558]      ste_t  *(*locate_by_ic)(char *);
[2559]      ste_t  *(*locate_next_by_ic)(ste_t *,char *);
[2560]      } ste_f_t;
[2561]      extern ste_f_t ste_f;
[2562]      static char *
[2563]      Locate(char *criteria)
[2564]      {
[2565]      static char gbuff[4096];
[2566]      char *answer=gbuff;
[2567]      char *name=NULL, *location=NULL;
[2568]      ste_t *ste;
[2569]      if(criteria && strlen(criteria) > 1)
        a.  if( *(criteria+strlen(criteria)-1) == '\n')
        b.  *(criteria+strlen(criteria)-1) = '\0';
[2570]      tds_get_nlpp(criteria,&name,&location,NULL,NULL,NULL,NULL);
[2571]      if(name == NULL || location == NULL) ste = ste_f.locate_by_ic(criteria);
[2572]      else ste = ste_f.locate_by_name_and_location(name,location);
[2573]      answer=gbuff;
[2574]      while(ste) {
        a.  ic_t *ic=ste->ic;
        b.  while( ic ) {
        c.  if( ic->public ) {
        d.  sprintf(answer,"%s=\"%s\" ", ic->field_name, ic->field_value);
        e.  answer=gbuff+strlen(gbuff);
        f.  }
        g.  ic=ic->next;
        h.  }
        i.  ste = ste_f.locate_next_by_ic(ste,criteria);
        j.  sprintf(answer,"\n");
        k.  answer=gbuff+strlen(gbuff);
[2575]      }
[2576]      return(gbuff);
[2577]      }
[2578]      static char *
[2579]      Delete(char *criteria)
[2580]      {
[2581]      return(NULL);
```

```
[2582]      }
[2583]      static char *
[2584]      Register(char *entry)
[2585]      {
[2586]      ste_t *ste = ste_f.constructor();
[2587]      char gbuff[2048];
[2588]      char *cp=entry;
[2589]      char *name=entry;
[2590]      char *value;
[2591]      static int regid;
[2592]      while( *cp ) {
        a.  while( *cp && isspace(*cp)) ++cp;
        b.  value=name=cp;
        c.  while( *value && *value != '=') ++value;
        d.  if( *value == '=')
        e.  {
        f.  int use_quote=0;
        g.  *value='\0';
        h.  ++value;
        i.  cp=value;
        j.  if( *value == "")
        k.  { use_quote=1; ++cp; ++value; }
        l.  while( *cp ) {
                i.  if( use_quote && (*cp == "")) break;
                ii. else if( use_quote==0 && isspace(*cp)) break;
                iii. ++cp;
        m.  }
        n.  if(*cp)  { *cp='\0'; ste_f.addic(ste,name,value); ++cp; }
        o.  else ste_f.addic(ste,name,value);
        p.  if( use_quote) ++cp;
        q.  }
[2593]      }
[2594]      sprintf(entry,"%d",regid);
[2595]      ++regid;
[2596]      return(entry);
[2597]      }
[2598]      typedef struct _tds_f {
[2599]      char *      (*query)(char *);
[2600]      char *      (*delete)(char *);
[2601]      char *  (*addentity)(char *);
[2602]      } tds_f_t;
```

[2603] tds_f_t tds_f = {

[2604] Locate,

[2605] Delete,

[2606] Register

[2607] };

[2608] // there are different ways to use this function.

[2609] // we can statically link it into an app,

[2610] // we can dynamically load it at run time

[2611] // we can invoke it from a generic front end loader

[2612] // it can be invoked by selecting on an icon having associated the generic front end loader

[2613] int tds(int argc, char **argv)

[2614] {

[2615] static char *gbuff;

[2616] static int gbuff_size;

[2617] int fd_in=0;

[2618] int fd_out=1;

[2619] int n;

[2620] char *command, *arg;

[2621] if(argv == NULL) { fd_in=argc; fd_out=argc; }

[2622] gbuff_size=2048;

[2623] gbuff=malloc(2048);

[2624] while(1) {

a. n=wait_read(fd_in,1000);
b. if(n == 0) continue;
c. n = peek(fd_in);
d. if(n < 0) break;
e. if(n > 0) {
f. if(n > (gbuff_size+1))

[2625] { if(gbuff) free(gbuff); gbuff=malloc(n+1); gbuff_size=n+1; }

a. memset(gbuff,'\0',gbuff_size);
b. n=readline(fd_in,gbuff,gbuff_size);
c. if(n==0) break;
d. arg=command=gbuff;
e. while(*arg && *arg != '\n' && ! isspace(*arg)) ++arg;
f. if(arg && isspace(*arg)) { *arg = '\0'; ++arg; }
g. if(strcmp(command,"register")==0) {
    i. char *tcid = tds_f.addentity(arg);
    ii. if(tcid) { write(fd_out,tcid,strlen(tcid)); write(fd_out,"\n",1); }
h. }
i. else if(strcmp(command,"query")==0) {
    i. char *entry;

ii. entry=tds_f.query(arg);

iii. if(entry)

[2626] { write(fd_out,entry,strlen(entry)); write(fd_out,"\n",1); }

a. }

b. else if(strcmp(command,"delete")==0) tds_f.delete(arg);

c. else if(strcmp(command,"QUIT")==0) break;

d. }

e. else break;

[2627] }

[2628] return (0);

[2629] }

[2630] Program Listing 14.0 process function - cps.c

[2631] #include <stdio.h>#include <fcntl.h>

[2632] extern int tid_size;

[2633] int

[2634] process(int fd, void *args, void *handle)

[2635] {

[2636] char *buffer, *sargs[3];

[2637] int p;

[2638] buffer=(char *)malloc(tid_size+5);

[2639] if((p = peek(fd)) != tid_size) return(-1);

[2640] memset(buffer,'\0',tid_size+5);

[2641] sprintf(buffer,"tid=");

[2642] read(fd,buffer+4,p);

[2643] if(peek(fd)) return(-1);

[2644] close(fd);

[2645] sargs[0] = "name=\"common directory service\"";

[2646] sargs[1] = buffer;

[2647] sargs[2] = '\0';

[2648] fd=open("payment_info",O_RDONLY);

[2649] dup2(fd,0);

[2650] gfel(2,sargs);

[2651] return(0);

[2652] }

[2653] Program Listing 14.1 process function - cps2.c

```
[2654] #include <stdio.h>#include <fcntl.h>extern int tid_size;extern int sid_size;int process(int fd, void *args, void *handle){   char *tidbuffer;
[2655] tidbuffer=(char *)malloc(tid_size+5);
[2656] if( peek(fd) != (tid_size+sid_size)) return(-1);
[2657] memset(tidbuffer,'\0',tid_size+5);
[2658] memset(sidbuffer,'\0',2048);
[2659] sprintf(tidbuffer,"tid=");
[2660] sprintf(sidbuffer,"query sid=");
[2661] read(fd,tidbuffer+4,tid_size);
[2662] read(fd,sidbuffer+10,sid_size);
[2663] if(peek(fd)) return(-1);
[2664] close(fd);
[2665] tds_query_p(sidbuffer, answer, 4096);
[2666] tds_get_nvpair(answer, "Service Provider", &sp);
[2667] if( ! sp ) { return(-1); }
[2668] fprintf(stdout,"Connection for Payment Information from %s\n",sp);
[2669] fprintf(stdout,"Accept? \n");
[2670] fscanf (stdin,"%s",answer);
[2671] if( strcmp(answer,"yes") != 0) return(-1);
[2672] sargs[0] = "name=\"common directory service\"";
[2673] sargs[1] = tidbuffer;
[2674] sargs[2] = '\0';
[2675] fd=open("payment_info",O_RDONLY);
[2676] dup2(fd,0);
[2677] gfel(2,sargs);
[2678] free(tidbuffer);
[2679] return(0);
[2680] }
```

[2681] Program Listing 14.2 process function - cps3.c

```
[2682] #include <stdio.h>#include <fcntl.h>extern int tid_size;extern int sid_size;int process(int fd, void *args, void *handle){   char *tidbuffer, sic
[2683] char *visa=NULL, *mc=NULL, mycardtype[512], mycardnumber[512];
[2684] char mycardexpires[512],  char buffer[2048];
[2685] int  fds[2];
[2686] tidbuffer=(char *)malloc(tid_size+5);
[2687] if( peek(fd) != (tid_size+sid_size)) return(-1);
```

```
[2688]  memset(tidbuffer,'\0',tid_size+5);
[2689]  memset(sidbuffer,'\0',2048);
[2690]  sprintf(tidbuffer,"tid=");
[2691]  sprintf(sidbuffer,"query sid=");
[2692]  read(fd,tidbuffer+4,tid_size);
[2693]  read(fd,sidbuffer+10,sid_size);
[2694]  if(peek(fd)) return(-1);
[2695]  close(fd);
[2696]  tds_query_p(sidbuffer, answer, 4096);
[2697]  tds_get_nvpair(answer, "American Express", &ae);
[2698]  tds_get_nvpair(answer, "Visa", &visa);
[2699]  tds_get_nvpair(answer, "MasterCard", &mc);
[2700]  sargs[0] = "name=\"common directory service\"";
[2701]  sargs[1] = tidbuffer;
[2702]  sargs[2] = '\0';
[2703]  fd=open("payment_info",O_RDONLY);
[2704]  pipe(fds);
[2705]  while(readline(fd,buffer,2048)) {
        a. if(ae) { ae=NULL; tds_getnvpair(buffer,"American Express",&ae);
        b. if(ae) { write(fds[1],buffer,strlen(buffer)); break; }
        c. }
        d. if( visa ) { visa=NULL; tds_getnvpair(buffer,"Visa",&visa);
        e. if(visa) {  write(fds[1],buffer,strlen(buffer)); break; }
        f. }
        g. if( mc ) { mc=NULL; tds_getnvpair(buffer,"MasterCard",&mc);
        h. if(mc) { write(fds[1],buffer,strlen(buffer)); break; }
        i. }
[2706]  }
[2707]  if( peek(fds[0])) { dup2(fds[0],0); gfel(2,sargs); }
[2708]  free(tidbuffer);
[2709]  close(fds[0]);
[2710]  close(fds[1]);
[2711]  return(0);
[2712]  }
```

[2713] Program Listing 14.3 process function - cps4.c

```
[2714]  #include <stdio.h>
[2715]  #include <fcntl.h>
```

```
[2716]  int process(int fd, void *args, void *handle)
[2717]  {
[2718]  char buffer[2048];
[2719]  char *t;
[2720]  int fdo, s;
[2721]  fdo=open((t=mktemp("/home/rs/rsXXXXXX")),O_WRONLY);
[2722]  while((s=read(fd,buffer,2048)) > 0 ) write(fdo,buffer,s);
[2723]  write(fd,t,strlen(t)); write(fd,"\n",1); close(fd);
[2724]  return(0);
[2725]  }
```

[2726] Program Listing 14.4 process function - cps5.c

```
[2727]  #include <stdio.h>
[2728]  #include <fcntl.h>
[2729]  int process(int fd, void *args, void *handle)
[2730]  {
[2731]  char fname[2048];
[2732]  char buffer[2048];
[2733]  char *t;
[2734]  int fdi, s;
[2735]  memset(fname,'\0',2048);
[2736]  readline(fd,fname,2048);
[2737]  if( (fdi=open(fname,O_RDONLY)) != -1) while((s=read(fdi,buffer,2048)) > 0 )
[2738]  write(fd,buffer,s);
[2739]  close(fdi);
[2740]  close(fd);
[2741]  return(0);
[2742]  }
```

[2743] Program Listing 15.0 stateful service - main.c

[2744] #include <stdio.h>int main(int argc, char *argv[]){ if(argc < 4) { fprintf(stderr,"service <protocol_ref> <poort> <service_name>\n");

a. exit(1);

```
[2745]  }
[2746]  tcp_accept(3, argv, argv[3]);
```

```
[2747]  return(0);
[2748]  }
```

[2749] Program Listing 15.1 stateful service - tcp_accept.c

```
[2750]  #include <stdio.h>#include <unistd.h>#include <stdlib.h>#include <sys/socket.h>#include <sys/types.h>#include
<sys/time.h>
[2751]  #include <fcntl.h>
[2752]  #include <errno.h>
[2753]  #include <netdb.h>
[2754]  #include <dlfcn.h>
[2755]  int tcp_accept(int argc, char **argv, char *service)
[2756]  {
[2757]  int         listenfd;
[2758]  int         connfd;
[2759]  socklen_t   addrlen=0, len;
[2760]  struct sockaddr  *cliaddr;
[2761]  time_t          ticks;
[2762]  char        buff[2048];
[2763]  void        *libhandle;
[2764]  int         (*function)(int,char *);
[2765]  if( (libhandle = dlopen("./libservices.so.1.0",RTLD_LAZY)) == NULL)
        a.  fprintf(stderr,"dlopen libservices.so failed errno=%d\n", errno);
[2766]  if( (function  = dlsym(libhandle, service)) == NULL) {
        a.  fprintf(stderr,"dlsym of function [%s] failed errno=%d\n", service, errno);
        b.  fprintf(stderr,"service not available\n");
        c.  exit(0);
[2767]  }
[2768]  if( argc == 2 ) listenfd = tcp_listen(NULL, argv[1], &addrlen);
[2769]  else listenfd = tcp_listen(argv[1], argv[2], &addrlen);
[2770]  cliaddr = malloc(addrlen);
[2771]  for( ; 1 ; ) {
        a.  len = addrlen;
        b.  connfd = accept(listenfd, cliaddr, &len );
        c.  printf("connection from: %s\n", sock_ntop(cliaddr,len));
        d.  if( function ) { (*function)(connfd,NULL); close(connfd); }
        e.  else  { ticks=time(NULL); sprintf(buff,"%.24s\n",ctime(&ticks));
        f.  write(connfd,buff,strlen(buff));
        g.  }
[2772]  }
```

[2773] }

[2774] Program Listing 15.2 stateful service - tcp_listen.c

```
[2775] #include <stdio.h>#include <unistd.h>#include <stdlib.h>#include <sys/types.h>#include <sys/time.h>
[2776] #include <fcntl.h>
[2777] #include <errno.h>
[2778] #include <netdb.h>
[2779] #include <dlfcn.h>
[2780] #if ! defined(LISTENQ)
[2781] #define LISTENQ     256
[2782] #endif
[2783] int tcp_listen(char *host, char *serv, socklen_t *addrlenp)
[2784] {
[2785] int listenfd, n;
[2786] const int on=1;
[2787] struct addrinfo hints, *res, *ressave;
[2788] void  *handle=NULL;
[2789] bzero( &hints, sizeof(struct addrinfo));
[2790] hints.ai_flags = AI_PASSIVE;
[2791] hints.ai_family= AF_UNSPEC;
[2792] hints.ai_socktype=SOCK_STREAM;
[2793] if((n=getaddrinfo_gamix(host, serv, &hints, &res)) != 0 ) {
a.  fprintf(stderr,"tcp_listen: getaddrinfo failed errno=%d\n", errno);
b.  exit(1);     }
[2794] ressave=res;
[2795] do { listenfd = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
[2796] if( listenfd < 0 ) continue;
[2797] setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
[2798] if( bind(listenfd, res->ai_addr, res->ai_addrlen) == 0 ) break;
[2799] close(listenfd);   } while((res=res->ai_next) != NULL );
[2800] if(res == NULL ) { fprintf(stderr,"tcp_listen: error for %s, %s\n", host,serv);
[2801] exit(1);     }
[2802] listen(listenfd,LISTENQ);
[2803] if(addrlenp) *addrlenp=res->ai_addrlen;
[2804] freeaddrinfo(ressave);
[2805] return(listenfd);
```

[2806] }

[2807] Program Listing 15.3 stateful service - getaddrinfo.c

[2808] /* portions of the code are derived from W. Richard Stevens implementation of getaddrinfo

1. See Unix Network Programming for details and variations

[2809] */

[2810] #include "services.h"#include <resolv.h>#include <ctype.h>#include <string.h>int getaddrinfo_gamix(const char *hostname, const char *servname, const struct addrinfo *hintsp, struct addrinfo **result)

[2811] {

[2812] int rc, error, nsearch;

[2813] char **ap, *canon=NULL;

[2814] struct hostent *hptr;

[2815] struct search search[3], *sptr;

[2816] struct addrinfo hints, *aihead, **aipnext;

[2817] #define error(e) { error = (e); goto bad; }

[2818] aihead=NULL;

[2819] aipnext=&aihead;

[2820] canon=NULL;

[2821] if(hintsp == NULL) { bzero(&hints, sizeof(hints)); hints.ai_family = AF_UNSPEC; }

[2822] else hints = *hintsp;

[2823] if((rc=ga_echeck(hostname, servname, hints.ai_flags, hints.ai_family, hints.ai_socktype, hints.ai_protocol)) != 0) error(rc);

[2824] if(hostname != NULL &&

a. (strcmp(hostname,"/local")==0 ||
b. strcmp(hostname,"/unix") ==0) &&
c. (servname != NULL && servname[0] == '/')) {
d. return(ga_unix(servname, &hints, result));

[2825] }

[2826] nsearch = ga_nsearch(hostname, &hints, &search[0]);

[2827] for(sptr = &search[0]; sptr < &search[nsearch]; sptr++) {

a. if(isdigit(sptr->host[0])) {
b. struct in_addr inaddr;
c. if(inet_pton(AF_INET,sptr->host, &inaddr) == 1) {
    i. if(hints.ai_family != AF_UNSPEC &&
    ii. hints.ai_family != AF_INET) error(EAI_ADDRFAMILY);
    iii. if(sptr->family != AF_INET) continue;
    iv. rc=ga_aistruct(&aipnext, &hints, &inaddr, AF_INET);
    v. if(rc != 0) error(rc);
    vi. continue;
d. }

```
e.    }
f.    if((isxdigit(sptr->host[0]) || sptr->host[0] == ':') &&
g.    (strchr(sptr->host,':') != NULL )) {
h.    struct in6_addr in6addr;
i.    if(inet_pton(AF_INET6, sptr->host, &in6addr)==1) {
          i.   if(hints.ai_family != AF_UNSPEC &&
          ii.  hints.ai_family != AF_INET6) error(EAI_ADDRFAMILY);
          iii. if( sptr->family != AF_INET6) continue;
          iv.  rc=ga_aistruct( &aipnext, &hints, &in6addr, AF_INET6);
          v.   if(rc != 0 ) error(rc);
          vi.  continue;
j.    }
k.    }
l.    if((_res.options & RES_INIT) == 0 ) res_init();
m.    if(nsearch == 2 ) {_res.options &= ~RES_USE_INET6;
[2828]    hptr = gethostbyname2(sptr->host, sptr->family);
a.    }
b.    else {
c.    if(sptr->family  == AF_INET6) _res.options |= RES_USE_INET6;
d.    else _res.options &= ~RES_USE_INET6;
e.    hptr = gethostbyname(sptr->host);
f.    }
g.    if( hptr == NULL ) { if(nsearch == 2 ) continue;
h.    switch(h_errno) {
i.    case HOST_NOT_FOUND: error(EAI_NONAME);
j.    case TRY_AGAIN: error(EAI_AGAIN);
k.    case NO_RECOVERY:error(EAI_FAIL);
l.    case NO_DATA: error(EAI_NODATA);
m.    default: error(EAI_NONAME);
n.    }
o.    }
p.    if(hints.ai_family != AF_UNSPEC && hints.ai_family != hptr->h_addrtype)
q.    error(EAI_ADDRFAMILY);
r.    if(hostname != NULL && hostname[0] != '\0' &&
s.    (hints.ai_flags & AI_CANONNAME) && canon==NULL) {
t.    if((canon=strdup(hptr->h_name)) == NULL)
          i.   error(EAI_MEMORY);
u.    }
v.    for(ap=hptr->h_addr_list; *ap != NULL; ap++ ) {
w.    rc=ga_aistruct(&aipnext, &hints, *ap, hptr->h_addrtype);
x.    if(rc != 0 ) error(rc);
y.    }
[2829]    }
[2830]    if( aihead == NULL ) error(EAI_NONAME);
[2831]    if(hostname != NULL && hostname[0] != '\0' && hints.ai_flags & AI_CANONNAME) {
a.    if( canon != NULL ) aihead->ai_canonname = canon;
b.    else {
c.    if( (aihead->ai_canonname = strdup(search[0].host))==NULL)
```

```
                i.  error(EAI_MEMORY);
        d.  }
[2832]      }

[2833]      if(servname != NULL && servname[0] != \0')
        a.  if((rc=ga_serv(aihead, &hints, servname)) != 0 ) error(rc);
[2834]      *result = aihead;
[2835]      return(0);
[2836]      bad:
[2837]      freeaddrinfo( aihead );
[2838]      return(error);
[2839]      }
```

[2840] Program Listing 16.1 - File SERVICES1 Service prototype table.

```
[2841]  %msg)flds)Description|Connectivity|Spid
[2842]  Payment Service|192.168.247:9999|1921653491024090
[2843]  Contact Information Service|192.168.247:9998|1921653491024092
[2844]  Directory Service|192.168.247:9997|1921653491024091
[2845]  URL Retrieval Service|192.168.247:9996|1921653491024075
```

[2846] Program Listing 16.2 - File SERVICES2 Service prototype table.

```
[2847]  %msg)flds)Description|InternetAddress|Port|Spid
[2848]  Payment Service|192.168.247|9999|1921653491024090
[2849]  Contact Information Service|192.168.247|9998|1921653491024092
[2850]  Directory Service|192.168.247|9997|1921653491024091
[2851]  URL Retrieval Service|elmer.gtlinc.com|9996|1921653491024075
```

[2852] Program Listing 16.3 - File SERVICES3 Service prototype table.

```
[2853]  %msg)flds)Description|InternetAddress|Port|Protocol|Spid
[2854]  Payment Service|192.168.247|9999|HTTP|1921653491024090
[2855]  Contact Information Service|192.168.247|9998|SOAP|1921653491024092
[2856]  Directory Service|192.168.247|9997|SOAP|1922653491024091
[2857]  URL Retrieval Service|elmer.gtlinc.com|9996|HTTP|1921653491024075
```

[2858] Program Listing 16.4 - Command line to generate data dictionary from prototype table

[2859]

```
$ DC-rcd SERVICES > rcd.SERVICES
```

[2860] Program Listing 16.5 - Generated Data Dictionary

[2861]

```
#{ RECORD_CLASS ( SERVICES )
a.  <Nbr_Per_Parent, ^0->^>
```

[2862]

```
#{ BIN_BUNCH
a.  #{ FILE ( SERVICES )
        i.   <Data_Interface, DC>
        ii.  <Unit_Sep, ^|^>
        iii. <Comment_Beg, ^%^>
        iv.  <Block_Beg, ^{^>
        v.   <Block_End, ^}^>
        vi.  <__Max_Rec_Len, 60>
        vii. <__Nbr_Of_Recs, 4> }#
```

[2863]

```
}#
```

[2864]

```
#{ BUFFER_BUNCH
a.  #{ BUFFER ( 1 )
b.  #{ KEY_BUNCH
c.  /** this is just a guess at a key **/
        i.   #{ KEY ( 1 )
                1.  <Field, Description>
        ii.  #{ INDEX_BUNCH
                1.  #{ INDEX ( 1 )
                2.  <Kind, btree> }#
        iii. }#
        iv.  }#
d.  }#
e.  #{ FORMAT
        i.   #{ FIELD ( Description )
                1.  <Datatype, STR>
                2.  <Size, ^*^>
                3.  <__Position, 1> }#
        ii.  #{ FIELD ( Connectivity )
                1.  <Datatype, STR>
                2.  <Size, ^*^>
                3.  <__Position, 2> }#
        iii. #{ FIELD ( Spid )
                1.  <Datatype, STR>
                2.  <Size, ^*^>
                3.  <__Position, 3> }#
f.  }#
g.  }#
```

[2865]

```
}#
```

[2866] }#

[2867] Program Listing 16.6 - Services2 prototype table

```
[2868] %msg)flds)Description|InetAddr|Port|Spsid|Name|Location|Id
[2869] Payment Service|192.168.247|9999|1931653491024090|||1931653491024090
[2870] Contact Service|192.168.247|9998|1729350331024097|||1931653491024091
[2871] Directory Service|192.168.247|9997|1729350331024097|||
[2872] MSQL sales report service|elmer.gtlinc.com|9997|1729350231024097|sr.qry||
[2873] UserService|tommy.gtlinc.com|9997|1729350231024097|userCheck|libservices.so.1.0
```

[2874] Program Listing 16.7 - Generated Data Dictionary for Services2

```
[2875] #{ RECORD_CLASS ( SERVICES2 )
[2876] <Nbr_Per_Parent, ^0->^>
[2877] <__Last_Validated_For_Rcd_UTC, ^1973-03-15@04:26:25^>
[2878] #{ BIN_BUNCH
[2879] #{ FILE ( SERVICES2 )
    a. <Data_Interface, DC>
    b. <Unit_Sep, ^|^>
    c. <Comment_Beg, ^%^>
    d. <Block_Beg, ^{^>
    e. <Block_End, ^}^>
    f. <__Max_Rec_Len, 93>
    g. <__Nbr_Of_Recs, 5>
    h. <__Size, 419> }#
[2880] }#
[2881] #{ BUFFER_BUNCH
[2882] #{ BUFFER ( 1 )
[2883] #{ KEY_BUNCH
    a. #{ KEY ( 1 )
    b. <Field, Description>
    c. #{ INDEX_BUNCH
    d. #{ INDEX ( 1 )
          i. <Kind, btree>
          ii. <__Btree_Height, 0>
          iii. <__Nbr_Of_Keys, 5>
          iv. <__Avg_Reach, 1.000>
          v. <__Nbr_Of_Keys_1, 5>
          vi. <__Avg_Reach_1, 1.000> }#
    e. }#
    f. }#
```

[2884] }#

[2885] #{ FORMAT

a. #{ FIELD (Description)
b. <Datatype, STR>
c. <Size, ^*^>
d. <__Position, 1> }#
e. #{ FIELD (Inetaddr)
f. <Datatype, STR>
g. <Size, ^*^>
h. <__Position, 2> }#
i. #{ FIELD (Port)
j. <Datatype, INT>
k. <Size, _long_>
l. <__Position, 3> }#
m. #{ FIELD (Spsid)
n. <Datatype, STR>
o. <Size, ^*^>
p. <__Position, 4> }#
q. #{ FIELD (Name)
r. <Datatype, STR>
s. <Nbr_Per_Parent, ^0->1^>
t. <Size, ^*^>
u. <__Position, 5> }#
v. #{ FIELD (Location)
w. <Datatype, STR>
x. <Nbr_Per_Parent, ^0->1^>
y. <Size, ^*^>
z. <__Position, 6> }#
aa. #{ FIELD (Id)
bb. <Datatype, STR>
cc. <Nbr_Per_Parent, ^0->1^>
dd. <Size, ^*^>
ee. <__Position, 7> }#

[2886] }#

[2887] }#

[2888] }#

[2889] }#

[2890] Program Listing 16.8 - Providers prototype table

[2891] %msg)flds)Name|Email|Id

[2892] Charles Northrup|cjn@gtlinc.com|1931653491024090

[2893] Frank Angel|franka@gtlinc.com|1931653491024091

[2894] Program Listing 16.9 - Providers generated data dictionary

[2895] #{ RECORD_CLASS (PROVIDERS)

[2896] <Nbr_Per_Parent, ^0->^>

[2897] <__Last_Validated_For_Rcd_UTC, ^1973-03-15@04:26:42^>

[2898] #{ BIN_BUNCH

[2899] #{ FILE (PROVIDERS)

a. <Data_Interface, DC>
b. <Unit_Sep, ^|^>
c. <Comment_Beg, ^%^>
d. <Block_Beg, ^{^>
e. <Block_End, ^}^>
f. <__Max_Rec_Len, 49>
g. <__Nbr_Of_Recs, 2>
h. <__Size, 120> }#

[2900] }#

[2901] #{ BUFFER_BUNCH

[2902] #{ BUFFER (1)

[2903] #{ KEY_BUNCH

a. #{ KEY (1)
b. <Field, Name>
c. #{ INDEX_BUNCH
d. #{ INDEX (1)
    i. <Kind, btree>
    ii. <__Btree_Height, 0>
    iii. <__Nbr_Of_Keys, 2>
    iv. <__Avg_Reach, 1.000>
    v. <__Nbr_Of_Keys_1, 2>
    vi. <__Avg_Reach_1, 1.000> }#
e. }#
f. }#

[2904] }#

[2905] #{ FORMAT

a. #{ FIELD (Name)
b. <Datatype, STR>
c. <Size, ^*^>
d. <__Position, 1> }#
e. #{ FIELD (Email)
f. <Datatype, STR>
g. <Size, ^*^>
h. <__Position, 2> }#
i. #{ FIELD (Id)
j. <Datatype, STR>
k. <Size, ^*^>
l. <__Position, 3> }#

[2906] }#

[2907] }#

[2908] }#

[2909] }#

[2910] Program Listing 16.10 - Cymbal instructions to insert record

```
[2911] do Insert_In_Services2();

[2912] global_defs:
[2913] define PROCEDURE transaction task :
[2914] Insert_In_Services2()
[2915] {
[2916] do Change so_that (
[2917] there_isa SERVICES2 where ( Description = "Recording Services"
    a. and Inetaddr = "porky.gtlinc.com"
    b. and Port = 9999
    c. and Spsid = "1725350231024097"
    d. and Id = "1931653491024090"
[2918] ));
[2919] }
```

[2920] Program Listing 16.11 - Cymbal instructions to report registration entry information

```
[2921] do Display with_format _table_ each [.description, .name, .id]
[2922] each_time(
[2923] there_is_a SERVICES2 where (
    a. Description = .description
    b. and Id = .id )
[2924] and there_is_a PROVIDERS where (
    a. Id= .id
    b. and Name = .name
[2925] )
[2926] )
```

[2927] Program Listing 16.12 - Global Definitions

```
[2928] global_defs:
[2929] define PROCEDURE transaction task :
[2930] Insert_In_Services2(STR .description,
    a. STR .inetaddr,
```

b. INT .port,
c. STR .spsid,
d. STR .id)

[2931] {

[2932] do Change so_that (

[2933] there_isa SERVICES2 where (Description = .description
a. and Inetaddr = .inetaddr
b. and Port = .port
c. and Spsid = .spsid
d. and Id = .id

[2934]));

[2935] }